DESIGNING PARSIMONIOUS REPRESENTATIONS OF THE MAXIMALLY PERMISSIVE DEADLOCK AVOIDANCE POLICY FOR COMPLEX RESOURCE ALLOCATION SYSTEMS THROUGH CLASSIFICATION THEORY

A Thesis
Presented to
The Academic Faculty

by

Ahmed Nazeem

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering, Georgia Institute of Technology

Georgia Institute of Technology
December 2012

DESIGNING PARSIMONIOUS REPRESENTATIONS OF THE MAXIMALLY PERMISSIVE DEADLOCK AVOIDANCE POLICY FOR COMPLEX RESOURCE ALLOCATION SYSTEMS THROUGH CLASSIFICATION THEORY

Approved by:

Professor Spyros Reveliotis, Advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Jeff Shamma
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Shabbir Ahmed
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Craig Tovey
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Stephane Lafortune
Electrical Engineering and Computer Science
University of Michigan

Date Approved: 5 July 2012

To the soul of my Father. To my beloved Mother.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Prof. Spyros Reveliotis for his guidance, care, patience, and supervision through the course of my graduate studies at Georgia Tech. Prof. Reveliotis' advice did not stop at research and writing manuscripts, I learnt from him the importance and the value of research and professional integrity. One simply could not wish for a better or friendlier supervisor.

I also want to thank Dr. Shabbir Ahmed, Dr. Stephane Lafortune, Dr. Jeff Shamma and Dr. Craig Tovey for their willingness to serve on my thesis committee. Furthermore, I would like to acknowledge the financial, academic and technical support of the School of Industrial and Systems Engineering and its staff, and particularly, Dr. Gary Parker for the support he provided to me as a new graduate student. Finally, I would like to thank the National Science Foundation which has supported part of my research.

The time I spent in Atlanta would have been dull if it were not for the great friends I met during my stay in Atlanta. To Aly Megahed, Ibrahim Ibrahim, Ahmed Mansy, Sherif Morad, Sami Majed, Alaa Elwany, Nader Metwalli, Faid Jradi, Akshay Gupte, Mashhour Solh, Hazem Nagy, Slim Ayadi, Tamer Gomaa, Abelkrim Khalif, and Farminder Anand: thank you very much for your friendship and for the good times we enjoyed together.

Last, but definitely not least, I would like to thank my mother and my sisters, Sara and Aya, for the love and support they have shown throughout these years. It has been a heartbreaking experience for me to be away from them all these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF NOMENCLATURE

$\chi(\mathbf{x}, \Lambda_{i'})$ An indicator that equals one iff $\mathbf{A}_{i'}\ \mathbf{x} \leq \mathbf{b}_{i'}$, where $(\mathbf{A}_{i'}, \mathbf{b}_{i'})$ are the inequalities defining the packing polyderon $\Lambda_{i'}$, p. 97.

η $max_{k=1}^{\mu} \eta_k$, p. 146.

η_{kl} The number of processing stages that request l units of the resource type R_k, p. 8.

η_k $max_{l=1}^{C_k} \eta_{lk}$, p. 140.

$\hat{S}_{r\bar{s}}^b$ The set of minimal boundary reachable unsafe states, p. 33.

$\hat{S}_{r\bar{s}}$ The set of minimal reachable unsafe states, p. 31.

$\hat{S}_{rd}$ The set of minimal reachable deadlock states, p. 31.

$\hat{S}_{rs}$ The set of maximal reachable safe states, p. 31.

$\kappa(\mathbf{x}, \Psi_{i^*})$ An indicator that equals zero iff $\mathbf{A}_{i^*}\ \mathbf{x} \geq \mathbf{b}_{i^*}$, where $(\mathbf{A}_{i^*}, \mathbf{b}_{i^*})$ are the inequalities defining the covering polyderon Ψ_{i^*}, p. 97.

λ_X $\lambda_X[q] \equiv max_{\mathbf{x} \in X} \mathbf{x}[q]$, $q = 1 \ldots \xi$, p. 138.

μ The number of the system resource types, p. 5.

Π A bijection that maps the elements of the dimension set L_P to the dimensions of subspace V_P, p. 33.

ρ $\sum_{k=1}^{\mu} \eta_k{}^{C_k}$, p. 142.

$\tau_{\mathbf{a}}$ A set of edges that emanate from state $\mathbf{a}$ and are known to lead to unsafe states, p. 151.

$\widehat{P(\hat{S}_{r\bar{s}}^b)}$ The set of simplified projected minimal boundary reachable unsafe states, p. 34.

$\widehat{P(\hat{S}_{rs})}$ The set of simplified projected maximal reachable safe states, p. 34.

ξ The number of processing stages supported by the considered RAS across the entire set of its process types, p. 6.

ζ The number of the system process types, p. 6.

$BlockEd[e]$ The set of minimal states at which the edge e is blocked, p. 139.

$BlockPs[q]$ The set of minimal states at which the processing stage q is blocked, p. 139.

C The system capacity function, p. 5.

C_i The capacity of the resource type R_i, p. 6.

$g(\mathbf{s})$ The set of enabled edges at state $\mathbf{s}$, p. 138.

h The number of the reader/writer resources, p. 172.

L The set of dimensions of the space V, p. 33.

L_0 The set of dimensions in L that satisfy Equation 2.2, p. 33.

L_P $L \backslash L_0$, p. 33.

$MinStR[k][l]$ The set of minimal states that allocate l units of the resource type R_k, p. 139.

$P(\hat{S}^b_{r\bar{s}})$ The set of projected minimal boundary reachable unsafe states, p. 33.

$P(\hat{S}_{rs})$ The set of projected maximal reachable safe states, p. 33.

$S^b_{r\bar{s}}$ The set of boundary reachable unsafe states, p. 27.

S_d The set of deadlock states, p. 13.

S_r The set of reachable states, p. 12.

S_s The set of safe states, p. 13.

$S_{\bar{r}}$ The set of unreachable states, p. 13.

$S_{\bar{s}}$ The set of unsafe states, p. 13.

$S_{r\bar{s}}$ The set of reachable unsafe states, p. 13.

S_{rd} The set of reachable deadlock states, p. 13.

S_{rs} The set of reachable safe states, p. 13.

V The ξ-dimensional vector space supporting the vector sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$, p. 33.

V_P The $|L_P|$-dimensional subspace supporting the projection of the vector sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ defined by removing the components of L_0, p. 33.

$\mathbf{s}.R_i$ The total number of units from resource R_i that are allocated at state $\mathbf{s}$, p. 138.

$\mathcal{A}$ The resource allocation function, p. 6.

$\mathcal{A}_{jk}[i]$ The number of resource units of resource type R_i necessary to support the execution of stage Ξ_{jk}, p. 6.

$\mathcal{D}(\Xi_{jk})$ The number of outgoing edges of the processing stage Ξ_{jk} in the graph $\mathcal{G}_j$, p. 8.

$\mathcal{G}_j$ A connect digraph that defines the sequential logic of process type J_j, p. 7.

$\mathcal{RW}$ The set of the reader/writer resources, p. 172.

$\mathcal{R}$ The set of the system resource types, p. 5.

covering polyhderon A polyhderon described in the form $\{\mathbf{x} \in \Re^n \mid \mathbf{Ax} \geq \mathbf{b},\ \mathbf{A} \geq \mathbf{0}$, and $\mathbf{b} \geq \mathbf{0}\}$, p. 97.

DAP Deadlock avoidance policy, p. 14.

FSA Finite state automaton, p. 10.

packing polyhedron A polyhderon described in the form $\{\mathbf{x} \in \Re^n \mid \mathbf{Ax} \leq \mathbf{b},\ \mathbf{A} \geq \mathbf{0}$, and $\mathbf{b} \geq \mathbf{0}\}$, p. 97.

RAS Resource allocation system, p. 5.

R/W-RAS Resource allocation system with reader/writer locks, p. 172.

SUMMARY

Most of the past research on the problem of deadlock avoidance for complex resource allocation systems (RAS) has acknowledged the fact that the computation of the maximally permissive (optimal) deadlock avoidance policy (DAP) possesses super-polynomial complexity for most RAS classes, and therefore, it has resorted to solutions that trade off maximal permissiveness for computational tractability. In this work, we distinguish between the *off-line* and the *on-line* computation that is required for the effective implementation of the maximally permissive DAP, and we seek to develop representations of this policy that will require *minimal* on-line computation. The particular representation that we adopt is that of a compact *classifier* that will effect the underlying dichotomy of the reachable state space into safe and unsafe subspaces. Through a series of reductions of the derived classification problem, we are also able to attain extensive reductions in the computational complexity of the off-line task of the construction of the sought classifier.

In a first study of the aforementioned problem, we restrict our attention to a particular RAS class that is motivated by an ongoing project called Gadara. This particular RAS class accepts the separation of the safe and unsafe subspaces of its instantiations through a set of linear inequalities. We propose design procedures that will construct a classifier employing the minimum possible number of linear inequalities, and we formally establish their "completeness", i.e., their ability to provide an effective classifier for every instance of the considered RAS class. We also offer heuristics that, if necessary, can alleviate the computational effort that is necessary for the construction of the sought classifier.

We extend the aforementioned results to encompass more general RAS classes, where the sought dichotomy might not be represented by a set of linear inequalities. To this end, we propose new parametric and non-parametric classification schemes for this more complex case, and establish formally their completeness. We also provide effective and computationally efficient procedures for the synthesis of the sought classifiers.

A bottleneck in the developments described above is defined by the fact that they presuppose the availability of the enumerations of the RAS safe and unsafe subspaces. To address this obstacle, we propose a novel approach for the deployment of the maximally permissive DAP for RAS, that is based on the identification and the efficient storage of a critical subset of states of the underlying RAS state space. In particular, the proposed algorithm provides those critical states, while avoiding the complete enumeration of the RAS state space.

Furthermore, we extend the existing theory on maximally permissive deadlock avoidance, so that it can handle RAS with reader/writer (R/W) locks. A key challenge that is posed by this new RAS class stems from the fact that the underlying state space is not necessarily finite. We effectively address this obstacle by taking advantage of special structure that exists in the set of unsafe states and enables a finite representation of this set through its minimal elements.

Finally, we would like to mention that numerical experimentation demonstrates the efficacy of the proposed approaches, and establishes their ability to support the deployment of maximally permissive DAP for RAS with very large structure and state spaces. To the best of our knowledge, these experiments also establish the ability of the proposed methodology to effectively compute tractable implementations of the maximally permissive DAP for problem instances significantly beyond the capacity of any other approach currently available in the literature. Moreover, this is the first work to address the RAS with R/W locks.

CHAPTER I

INTRODUCTION AND LITERATURE SURVEY

In the last few decades, there has been an increasing demand for flexible automation. This trend manifests itself in various domains like flexible automated production environments, transportation systems, workflow management, material handling, and telecommunication. In automated production systems, common equipment may be engaged in the production of different items, and the production system must be managed in a way that provides each workpiece with the necessary processing and handling. In railway systems, the vehicles connecting different pairs of locations share various track segments, and their access to these tracks must be managed to avoid the collision between the vehicles. In material handling systems, like the Automatic Guided Vehicle (AGV) systems, the traffic dynamics resemble that of the railway systems mentioned above. In particular, each vehicle trip should be managed so that each vehicle finishes its task without colliding with the other vehicles. More recently, in the emerging paradigm of multi-threaded programming, different threads operate on common data sets, and their execution must be managed in a way that avoids having inconsistencies in the shared data. The significant common characteristic among all the aforementioned domains is that the underlying operations can be abstracted to a set of processes that contest for the engagement of the system resources. Furthermore, all these systems exhibit a high degree of resource sharing in order to increase their flexibility. Hence, the notion of the "*sequential resource allocation systems (RAS)*" [72] has been formally defined in the literature as an abstraction of the structure of the resource allocation function taking place in all the aforementioned applications, and the resultant dynamics have been further modeled

and analyzed through representations and techniques borrowed from (qualitative) Discrete Event Systems (DES) theory [9].

In general, the control –or the real-time management– of sequential RAS comprises two dimensions: logically/behaviorally-oriented control and performance-oriented control, i.e., scheduling. The logically-oriented control addresses the behavioral correctness of the system; i.e., given some behavioral specifications, its goal is to prevent the occurrence of events that violate the given specifications. A typical behavioral property sought in such systems is the *non-blocking* behavior, i.e., each activated process should be able to proceed towards completion without experiencing any permanent stalling and the further need for external (human) intervention. In general, the behavior attained by applying the necessary logically-oriented control for addressing the posed specification is called the *admissible* behavior (under the given specification). On the other hand, the performance-oriented control addresses the problem of biasing the admissible behavior to satisfy given performance objectives. This boils downs to the typical sequencing and scheduling problems addressed in the OR literature. A control framework for the real-time management of sequential RAS is illustrated in Figure 1.1.

Our research addresses the construction of a logically-oriented controller to enforce the non-blocking behavior of the underlying RAS. This problem is typically known in the literature as the *Deadlock Avoidance Problem* for sequential resource allocation systems. The study of the deadlock avoidance problem was initiated in the late 60's and early 70's, in the context of the computing technologies that were emerging at that time [31, 30, 14, 32]. Some of the main contributions of that era were (i) the formalization of the concept of deadlock and of the resource allocation dynamics that lead to its formation by means of graph-theoretic concepts and structures, and (ii) the identification of off-line structural conditions and on-line resource allocation policies that would guarantee the deadlock-free operation of the underlying system.

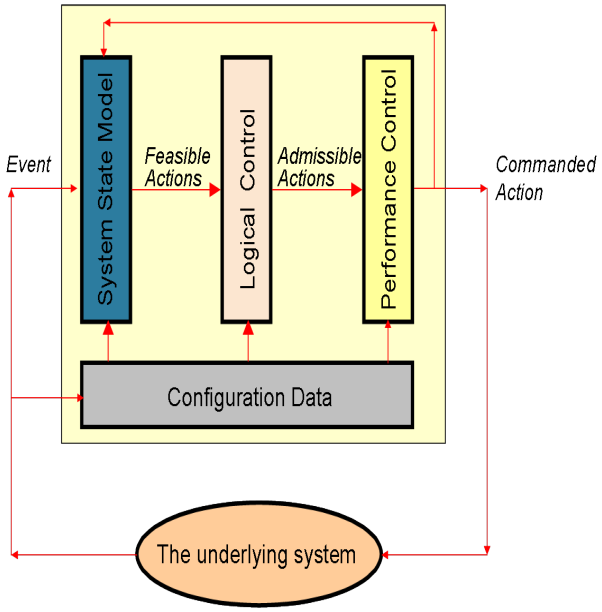


Figure 1.1: A framework for the real-time management of sequential RAS [72]

The design of the resource allocation processes so that they do not give rise to any circular waiting patterns is an example of the aforementioned structural conditions, while Banker's algorithm [21] is the best known deadlock avoidance policy (DAP) of that era.

The problem of deadlock avoidance was subsequently revived in the late 80's / early 90's, primarily in the context of the resource allocation taking place in flexibly automated production systems and intelligent transportation systems. The defining characteristics of these new studies were (i) the better specificity, tractability, and predictability of the underlying resource allocation processes with respect to their resource allocation requests, and (ii) the employment of the simultaneously emerging qualitative Discrete Event Systems (DES) theory [68, 9] as a powerful and rigorous base for modeling, analyzing and eventually controlling the considered RAS dynamics. The combination of these two effects has led to a more profound understanding of the process of deadlock formation and of the RAS structural attributes that affect this process, under various DES-based representations.

Hence, it is currently known that the aforementioned deadlock avoidance problem can be characterized in the classical Ramadge & Wonham Supervisory Control (R&W SC) framework [68, 9] in a straightforward manner, by (i) expressing the underlying resource allocation dynamics through a Finite State Automaton (FSA), and (ii) requesting the confinement of the RAS behavior to the subspace of this FSA that is defined by its maximal strongly connected component that contains the system state $\mathbf{s}_0$ where the RAS is idle and empty of any jobs. In fact, this characterization of the problem and its solution establishes also a notion of optimality for the considered problem, since the resulting policy prevents the formation of deadlock while retaining the maximum possible behavioral latitude for the underlying RAS.

However, when it comes to the implementation of the control function, the standard approach of R&W SC theory still employs the FSA representation of the controlled system behavior under the target policy, and for many practical RAS configurations, the explicit storage and on-line parsing of this information is of prohibitive computational cost due to the enormous size of the involved state spaces. Alternatively, one can consider a one-step lookahead control scheme that would seek the "real-time" – or the "on-line" – assessment of the co-accessibility of any given RAS state to the empty state $\mathbf{s}_0$, a property that is otherwise known as the state *"safety"*, in the relevant terminology; but this approach is also computationally challenged, since, for most RAS classes, the assessment of state safety is an NP-complete problem [3, 29, 41].

Therefore, the implementation of the optimal, or more generally, a highly permissive DAP for the sequential RAS that are encountered in contemporary applications remains an open and challenging task. On the other hand, the industrial practice is becoming increasingly demanding for solutions that are highly permissive, and at the same time, computationally efficient. As a case in point, let us consider our experience with the "Gadara" project, that has been a major motivation for this work. The

Gadara project was initiated at the University of Michigan, in collaboration with HP Labs, and its goal is to develop a software tool that will automatically take a multi-threaded deadlock-prone program as input, and instrument it with appropriate control logic so that the resultant code is deadlock-free. Hence, the entire problem addressed by Gadara reduces to the design and the deployment of a DAP that will manage the allocation of the "locks" shared by the concurrently executing processes (i.e., threads) in the context of the considered program [89, 90, 91, 38]. Furthermore, in the Gadara context, the deployed DAP must ensure the safe and live execution of the underlying processes in a way that (i) imposes the minimal necessary restriction on the execution of these processes, and (ii) causes the minimum possible computational "overhead" for the underlying system. Requirement (i) implies that the maximal permissiveness of the applied control logic is of paramount importance in the considered application context. Requirement (ii) introduces the additional notion of *"structural minimality"* for the derived supervisors.

The rest of this chapter will evolve as follows: Section 1.1 introduces formally the concept of the resource allocation systems employed in this work. Next, Sections 1.2 and 1.3 present the two main modeling frameworks used in the literature for analyzing the behavior of the resource allocation systems, Finite State Automata and Petri nets. Finally, Section 1.4 discusses the current state of art of the deadlock avoidance problem.

1.1 Resource Allocation Systems

Definition 1.1 *[72] A* (sequential) resource allocation system (RAS) *is defined as a 4-tuple* $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$ *where:*

1. $\mathcal{R} = \{R_1, \ldots, R_\mu\}$ *is the set of the system* resource types*, and* μ *is the cardinality of this set.*

2. $C : \mathcal{R} \rightarrow \mathbb{Z}^+$ *is the system* capacity *function, with* $C(R_i) \equiv C_i$ *characterizing the number of identical units from resource type* R_i *that are available in the system. Resources are considered to be* reusable*, i.e., they are engaged by the various processes according to an allocation/de-allocation cycle, and each such cycle does not affect their functional status or subsequent availability.*

3. $\mathcal{P} = \{J_1, \ldots, J_\zeta\}$ *is the set of the system* process types *supported by the considered system configuration, and* ζ *is the cardinality of this set. Each process type* J_j *is a composite element itself; in particular,* $J_j = < \mathcal{S}_j, \mathcal{G}_j >$*, where:*

   (a) $\mathcal{S}_j = \{\Xi_{j1}, \ldots, \Xi_{j,\varpi(j)}\}$ *is the set of* processing stages *involved in the definition of process type* J_j*, and*

   (b) $\mathcal{G}_j$ *is a data structure that defines the sequential logic over the set of processing stages* $\mathcal{S}_j$*, that governs the execution of any process instance of type* J_j*.*

4. $\mathcal{A} : \bigcup_{j=1}^{\zeta} \mathcal{S}_j \rightarrow \prod_{i=1}^{\mu} \{0, \ldots, C_i\}$ *is the* resource allocation function*, which associates every processing stage* Ξ_{jk} *with a* resource allocation request $\mathcal{A}(j,k) \equiv \mathcal{A}_{jk}$*. More specifically, each* $\mathcal{A}_{jk}$ *is an* μ*-dimensional vector, with its* i*-th component indicating the number of resource units of resource type* R_i *necessary to support the execution of stage* Ξ_{jk}*. Obviously, in a well-defined RAS,* $\mathcal{A}_{jk}[i] \leq C_i$*,* $\forall j,k,i$*. Also, it is assumed that* $\mathcal{A}_{jk} \neq \mathbf{0}$*, i.e., every processing stage requires at least one resource unit for its execution.*

For complexity considerations, we also define the quantity $|\Phi| \equiv |\mathcal{R}| + |\bigcup_{j=1}^{\zeta} \mathcal{S}_j| + \sum_{i=1}^{\mu} C_i$ as the *"size"* of RAS Φ. Furthermore, for notational convenience, we shall set $\xi \equiv \sum_{j=1}^{\zeta} |\mathcal{S}_j|$; i.e., ξ denotes the number of distinct processing stages supported by the considered RAS, across the entire set of its process types. Finally the various processing stages Ξ_{jk}, $j = 1, \ldots, \zeta$, $k = 1, \ldots, \varpi(j)$, will frequently be considered in the

context of a total ordering imposed on the set $\bigcup_{j=1}^{\zeta} \mathcal{S}_j$; in that case, the processing stages themselves and their corresponding attributes will be indexed by a single index q that runs over the set $\{1,\ldots,\xi\}$, and indicates the position of the processing stage in the considered total order.

Moreover, the RAS class to be considered in this work satisfies the following two assumptions:

Assumption 1.1 *In the considered RAS, the data structure $\mathcal{G}_j$ that defines the sequential logic of process type J_j, $j=1,\ldots,\zeta$, corresponds to a connected digraph $(\mathcal{V}_j,\mathcal{E}_j)$, where the graph node set $\mathcal{V}_j$ is in one-to-one correspondence with the processing stage set, $\mathcal{S}_j$. Furthermore, there are two subsets $\mathcal{V}_j^{\nearrow}$ and $\mathcal{V}_j^{\searrow}$ of $\mathcal{V}_j$ respectively defining the sets of initiating and terminating processing stages for process type J_j. The connectivity of digraph $\mathcal{G}_j$ is such that every node $v \in \mathcal{V}_j$ is accessible from the node set $\mathcal{V}_j^{\nearrow}$, and is co-accessible– i.e., can reach– to the node set $\mathcal{V}_j^{\searrow}$. Finally, any directed path of $\mathcal{G}_j$ leading from a node of $\mathcal{V}_j^{\nearrow}$ to a node of $\mathcal{V}_j^{\searrow}$ constitutes a complete execution sequence – or a "route" – for process type J_j.*

Assumption 1.2 *In the considered RAS, the resource allocation requests $\mathcal{A}_{jk}$, $j=1,\ldots,\zeta$, $k=1,\ldots,\varpi(j)$, are "conjunctive", i.e., a processing stage Ξ_{jk} can request an arbitrary nonempty subset of the system resources for its execution. Furthermore, a process instance executing processing stage Ξ_{jk} will be able to advance to a successor processing stage $\Xi_{j,k+1}$, only after it is allocated the resource differential $(A_{j,k+1}-A_{jk})^{+}$; and it is only upon this advancement that the process will release the resource units $|(A_{j,k+1}-A_{jk})^{-}|$, that are not needed anymore.*

Given an edge $e \in \mathcal{G}_j$ linking Ξ_{jk} to $\Xi_{j,k+1}$, we define $e.src \equiv \Xi_{jk}$ and $e.dst \equiv \Xi_{j,k+1}$, i.e., $e.src$ and $e.dst$ denote respectively the source and the destination nodes of edge e. Also, in the following, we shall use the notation $\mathcal{G}$ to refer to the "union" of graphs $\mathcal{G}_j$, $j=1,\ldots,\zeta$, i.e., $\mathcal{G} \equiv (\mathcal{V},\mathcal{E})$, with $\mathcal{V}=\cup_{j=1}^{\zeta}\mathcal{V}_j$ and $\mathcal{E}=\cup_{j=1}^{\zeta}\mathcal{E}_j$. Also, the number of

Table 1.1: A RAS taxonomy

Based on the structure of the Process Sequential Logic	**Based on the structure of the Resource Requirement Vectors**
Linear: Each process is defined by a linear sequence of stages	**Single-Unit:** Each stage requires a single unit from a single resource
Disjunctive: A number of alternative process plans encoded by a connected digraph	**Single-Type:** Each stage requires an arbitrary number of units, but all from a single resource
Merge-Split: Each process is a fork-join network	
Complex: A combination of the above behaviors	**Conjunctive:** An arbitrary number of units from different resources

outgoing edges of Ξ_{jk} shall be denoted by $\mathcal{D}(\Xi_{jk})$. Finally, we shall let η_{kl}, $l = 1 : C_k$, denote the number of processing stages that request l unit of the resource type R_k.

A more general definition of the RAS concept is provided in [72]. The basic differences between Definition 1.1 and the RAS definition of [72] can be summarized as follows: First of all, the complete definition of a RAS, according to [72], involves an additional component that characterizes the time-based – or *quantitative* – dynamics of the RAS; but this component is not relevant in the modeling and analysis to be pursued in the following developments, and therefore, it is omitted. Furthermore, the process-defining logic supported by Definition 1.1 encompasses the operational feature of routing flexibility, but it excludes the possibility of merging and splitting operations. More generally, one can classify the various instantiations of the RAS concept into a taxonomy that is defined on the basis of (i) the structural characteristics of the sequential logic that defines the process routes, and (ii) the complexity of the resource allocation function as expressed by the resource requests that are posed by the various processing stages. Then, the main RAS classes that are identified and supported by the RAS definition of [72] are provided in Table 1.1. The reader should notice that the RAS defined in Definition 1.1 above, essentially corresponds to the *Disjunctive/Conjunctive (D/C-)* RAS class in the taxonomy of Table 1.1.

Table 1.2: The RAS considered in Example 1.1

Resource Types	$\{R_1, R_2, R_3\}$
Resource Capacities	$C(R_1) = C(R_2) = C(R_3) = 1$
Process Types	$\{J_1, J_2\}$
Process Routes	$\mathcal{G}_1: \quad R_1 \rightarrow R_2 \rightarrow R_3$
	$\mathcal{G}_2: \quad R_3 \rightarrow R_2 \rightarrow R_1$

Example 1.1 We demonstrate the RAS concept implied by Definition 1.1 by introducing a particular RAS instance that will also provide an expository base for the subsequent discussion. The RAS depicted in Table 1.1 consists of three resource types $R_1,\ R_2$, and R_3, each of unit capacity, and two process types J_1 and J_2. The sequential logic corresponding to each of these processes has a simple linear structure involving three stages. The resource allocation function, $\mathcal{A}$, of this RAS can be derived from the information on the process routes provided in the table. For example, the processing stage Ξ_{12} needs only one unit of the resource type R_2 to support its execution. Hence, $\mathcal{A}_{12}[1] = \mathcal{A}_{12}[3] = 0$, and $\mathcal{A}_{12}[2] = 1$. $\square$

1.1.1 Gadara RAS

The next definition specializes the RAS abstraction that was defined in the previous pages to the resource allocation dynamics addressed by the Gadara project.

Definition 1.2 *A RAS instance Φ_g is called a* Gadara *RAS if (i) it satisfies Assumptions 1.1 and 1.2, and (ii) $C_i = 1,\ \forall i = 1, \ldots, \mu$.*

The resource types of the Gadara RAS model the mutual exclusion locks acquired by contesting threads. According to the typically employed locking synchronization mechanism, at any point of time, at most one thread can acquire a given lock. This requirement implies the single-unit capacity constraint introduced by the second condition of Definition 1.2. Since item C has its value explicitly fixed by Definition 1.2,

it will be dropped from the specification of the Gadara RAS, and consequently, Φ_g will be defined by the 3-tuple $\langle \mathcal{R}, \mathcal{P}, \mathcal{A} \rangle$.

1.1.2 RAS with Reader/Writer locks

Reader-writer synchronization, as introduced by [17], relaxes the constraints of mutual exclusion to permit the inspection of a shared resource simultaneously by more than one process, as long as none of them changes its value. Thus, multiple processes can read from a shared resource simultaneously, but a process can write to a shared resource only if no other process is writing to, or reading from, this resource. We shall characterize this effect by saying that the R/W locks, when perceived as resources, work in two modes: (i) a writing mode where the resource capacity is equal to one, and (ii) a reading mode where the capacity is infinite. We shall refer to a RAS containing R/W locks as a *"R/W-RAS"*.

1.2 FSA-based modeling and analysis of RAS behavior

The Finite State Automaton is a simple and straightforward formal representation for the RAS behavior[1]. The general definition of the FSA concept is provided in Appendix A. The dynamics of the RAS $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$, introduced in the previous section, can be formally described by a *Deterministic Finite State Automaton (DFSA)* ([9]), $G(\Phi) = \langle S, E, f, \Gamma, \mathbf{s}_0, S_M \rangle$, that is defined as follows:

1. The *state set* S consists of ξ-dimensional vectors $\mathbf{s}$. The components $\mathbf{s}[q]$, $q = 1, \ldots, \xi$, of $\mathbf{s}$ are in one-to-one correspondence with the RAS processing stages, and they indicate the number of process instances executing the corresponding stage in the RAS state modeled by $\mathbf{s}$. Hence, S consists of all the vectors

[1]While the FSA is capable of representing the behavior of the RAS class introduced in Def. 1.1, it is not capable of representing the behavior of the R/W-RAS due to the potential infiniteness of its underlying state space. This difficulty will be addressed in Chapter 7.

$\mathbf{s} \in (\mathbb{Z}_0{}^+)^\xi$ that further satisfy

$$\forall i = 1, \ldots, \mu, \quad \sum_{q=1}^{\xi} \mathbf{s}[q] \cdot \mathcal{A}(\Xi_q)[i] \leq C_i \tag{1.1}$$

where $\mathcal{A}(\Xi_q)[i]$ denotes the allocation request for resource R_i that is posed by stage Ξ_q. [2]

2. The *event set* E is the union of the disjoint event sets $E^{\nearrow}$, $\bar{E}$ and $E^{\searrow}$, where:

   (a) $E^{\nearrow} = \{e_{rp} : r = 0,\ \Xi_p \in \bigcup_{j=1}^{\zeta} \mathcal{V}_j^{\nearrow}\}$, i.e., event e_{rp} represents the *loading* of a new process instance that starts from stage Ξ_p.

   (b) $\bar{E} = \{e_{rp} : \exists j \in 1, \ldots, \zeta \text{ s.t. } \Xi_p \text{ is a successor of } \Xi_r \text{ in digraph } \mathcal{G}_j\}$, i.e., e_{rp} represents the *advancement* of a process instance executing stage Ξ_r to a successor stage Ξ_p.

   (c) $E^{\searrow} = \{e_{rp} : \Xi_r \in \bigcup_{j=1}^{\zeta} \mathcal{V}_j^{\searrow},\ p = 0\}$, i.e, e_{rp} represents the *unloading* of a finished process instance after executing its last stage Ξ_r.

3. The *state transition function* $f : S \times E \to S$ is defined by $\mathbf{s}' = f(\mathbf{s}, e_{rp})$, where the components $\mathbf{s}'[q]$ of the resulting state $\mathbf{s}'$ are given by:

$$\mathbf{s}'[q] = \begin{cases} \mathbf{s}[q] - 1 & \text{if} \quad q = r \\ \mathbf{s}[q] + 1 & \text{if} \quad q = p \\ \mathbf{s}[q] & \text{o.w.} \end{cases}$$

   Furthermore, $f(\mathbf{s}, e_{rp})$ is a *partial* function defined only if the resulting state $\mathbf{s}' \in S$.

[2] Following standard practice in DES literature (cf., for instance, the relevant definition in page 8 of [9]), in the rest of this document we will frequently use the terms "space" and "subspace" in order to refer to set S and its various subsets considered in this work. We want to emphasize, however, that S and its various subsets involved in this work are *not* vector spaces in the sense that this term is used in linear algebra since they are not closed to vector addition and scalar multiplication.

4. The *feasible event function* $\Gamma: S \to 2^E$ collects, for each state $\mathbf{s} \in S$, the set of events $e \in E$ for which the transition function $f(\mathbf{s}, e)$ is defined (i.e., the resulting state $\mathbf{s}'$ belongs in S). [3]

5. The *initial state* $\mathbf{s}_0$ is set equal to $\mathbf{0}$.

6. The *set of marked states* S_M is the singleton $\{\mathbf{s}_0\}$.

Let $\tilde{f}$ denote the natural extension of the state transition function f to $S \times E^*$; i.e., for any $\mathbf{s} \in S$ and the empty event string ϵ, $\tilde{f}(\mathbf{s}, \epsilon) = \mathbf{s}$, while for any $\mathbf{s} \in S$, $\sigma \in E^*$ and $e \in E$, $\tilde{f}(\mathbf{s}, \sigma e) = f(\tilde{f}(\mathbf{s}, \sigma), e)$. Also, it is implicitly assumed that $\tilde{f}(\mathbf{s}, \sigma e)$ is undefined if any of the one-step transitions that are involved in the right-hand-side recursion are undefined.

Given a RAS state, we can infer the corresponding resource allocation of the RAS resource types by means of the resource allocation function $\mathcal{A}$. We further notice that for the RAS classes that are encompassed by Definition 1.1, the finiteness of the RAS state space defined above results from that fact that every processing stage requires at least one resource unit for its execution, and each resource type has finite capacity.

Of particular significance in the FSA-based representation of the RAS dynamics is the empty state $\mathbf{s}_0$, since it represents the initial (idle) state and the state that results from the complete processing of a given set of activated process instances. The behavior of RAS Φ is modeled by the *language* $L(G)$ *generated* by DFSA $G(\Phi)$, i.e., by all strings $\sigma \in E^*$ such that $\tilde{f}(\mathbf{s}_0, \sigma)$ is defined. States that can be reached from $\mathbf{s}_0$ through a sequence of feasible events constitute the reachable subspace of the FSA, S_r. More formally, the *reachable subspace* of $G(\Phi)$ is the subset S_r of S defined as follows:

[3] The reader should notice that in the considered FSA, the definition of Γ is immediately induced from the information conveyed by the definition of the state transition function in item #3 above, and therefore, this element is introduced only for completeness and consistency with the general definition that is provided in Appendix A.

$$S_r \equiv \{\mathbf{s} \in S : \exists \sigma \in L(G) \text{ s.t. } \tilde{f}(\mathbf{s}_0, \sigma) = \mathbf{s}\} \tag{1.2}$$

We also define the *safe subspace* of $G(\Phi)$, S_s, by:

$$S_s \equiv \{\mathbf{s} \in S : \exists \sigma \in E^* \text{ s.t. } \tilde{f}(\mathbf{s}, \sigma) = \mathbf{s}_0\} \tag{1.3}$$

S_s contains those states of S that are co-accessible to the marked state $\mathbf{s}_0$, i.e., those states from which $\mathbf{s}_0$ can be reached. In the following, we shall denote the complements of S_r and S_s with respect to S by $S_{\bar{r}}$ and $S_{\bar{s}}$, and we shall refer to them as the *unreachable* and *unsafe* subspaces. Moreover, S_{xy}, $x \in \{r, \bar{r}\}$, $y \in \{s, \bar{s}\}$, will denote the intersection of the corresponding sets S_x and S_y. Hence, S_{rs} (resp., $S_{r\bar{s}}$) shall denote the set of reachable safe (resp., unsafe) states.

The RAS unsafety characterized in the previous paragraph results from the formation of RAS *deadlocks*, i.e., RAS states where a subset of the running processes are entangled in a circular waiting pattern for resources that are held by other processes in this set, blocking, thus, the advancement of each other in a permanent manner. The RAS unsafe states are essentially those RAS states from which the formation of deadlock is unavoidable. In the following, the set of deadlock states will be denoted by S_d, while S_{rd} will denote the set of reachable deadlock states. Finally, it is clear from the above that $S_d \subseteq S_{\bar{s}}$ and $S_{rd} \subseteq S_{r\bar{s}}$.

Example 1.1 (cont.) Figure 1.2 depicts the FSA that corresponds to the reachable subspace of the RAS defined in Table 1.1. This RAS evolves into states from which $\mathbf{s}_0$ is not accessible any more. This RAS behavior is characterized as *blocking*, and as explained above, it is the result of potential deadlocks that might arise among the running processes. In Figure 1.2, states q^{16}, q^{17}, q^{18}, and q^{19} are deadlock states. On the other hand, it is also interesting to notice that state q^{15} is not a deadlock state; however, from this state, the system will evolve inevitably to a deadlock. States with this property are called *deadlock-free unsafe* states. In Figure 1.2, $S_{rs} \equiv \{q^1, \ldots, q^{14}\}$,

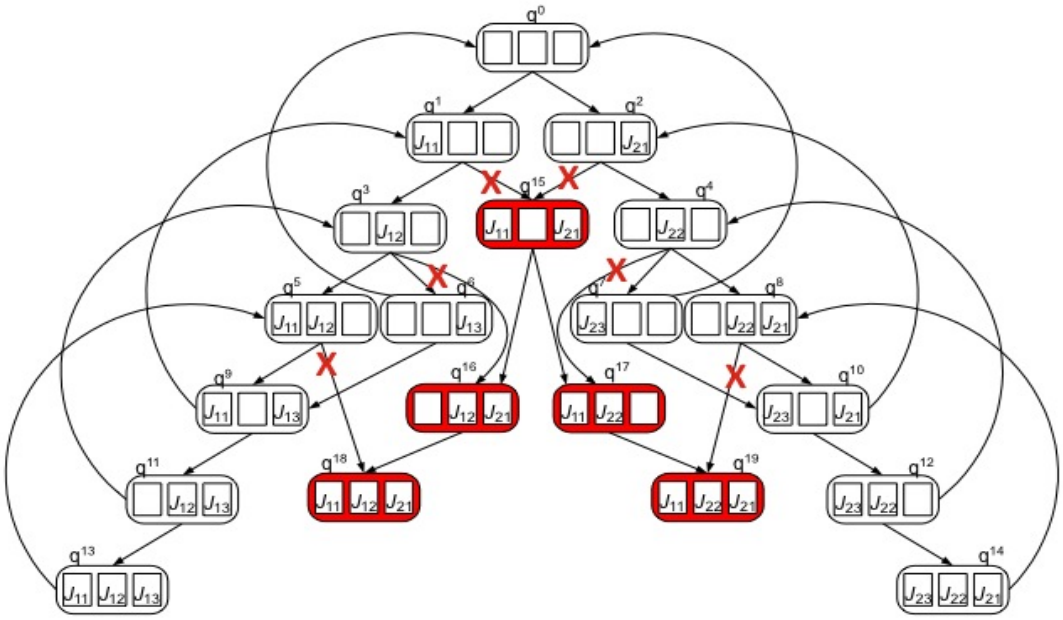


Figure 1.2: The finite state automaton modeling the reachable sub-space of the RAS defined in Table 1.1. In the adopted state representation, the three inner cells indicate the status of resources R_1, R_2, and R_3, in this order. Empty cells imply a free resource. Cells corresponding to allocated resources are annotated by the processing stage of the process instance that holds the corresponding resource. States depicted in red are the unsafe states that must be eliminated from the system behavior through the employment of a deadlock avoidance policy. In particular, the maximally permissive DAP must recognize and block the transitions indicated by the red crossings in the figure.

$S_{r\bar{s}} \equiv \{q^{15}, q^{16}, q^{17}, q^{18}, q^{19}\}$, and $S_{rd} \equiv \{q^{16}, q^{17}, q^{18}, q^{19}\}$.

The target behavior of $G(\Phi)$ and the structure of the maximally permissive deadlock avoidance policy The maximally permissive deadlock avoidance policy (DAP) is the policy that eliminates all the unsafe states from the RAS behavior, while retaining all the safe states. More formally, the target behavior of RAS Φ is expressed by the *marked language* $L_m(G)$, which is defined by means of the marked

state $\mathbf{s}_0$, as follows:

$$L_m(G) \equiv \{\sigma \in L(G) : \tilde{f}(\mathbf{s}_0, \sigma) = \mathbf{s}_0\} \tag{1.4}$$

Equation 1.4, when combined with all the previous definitions, further implies that the set of states that are accessible under $L_m(G)$ is exactly equal to S_{rs}. Hence, we have the following definition of the *maximally permissive deadlock avoidance policy (DAP)* for the considered RAS:

Definition 1.3 *The* maximally permissive deadlock avoidance policy (DAP) *for any instantiation* Φ *from the considered RAS class of Definition 1.1 is a supervisory control policy that admits a feasible transition* $\mathbf{s}' = f(\mathbf{s}, e_{rp})$ *of the underlying DFSA* $G(\Phi)$ iff $\mathbf{s}' \in S_s$.

In other words, starting from state $\mathbf{s}_0$, a maximally permissive DAP must allow a system-enabled transition to a next state $\mathbf{s}'$ if and only if (*iff*) $\mathbf{s}'$ belongs to S_s. This characterization of the maximally permissive DAP ensures its uniqueness for any given RAS instantiation. It also implies that the policy can be effectively implemented through any mechanism that recognizes and rejects the unsafe states that are accessible through one-step transitions from S_{rs}.

1.3 Petri Net-based modeling

The Petri net is the other major modeling framework, besides FSA, employed in the logical analysis of Discrete Event Systems (DES) [54, 9]. In its basic definition, a Petri net (PN) is a weighted bipartite graph with two node classes, P and T, respectively characterized as *places* and *transitions*. The underlying system state is expressed by the PN marking, $M : P \to Z_0^+$, i.e., a mapping of the net places to the set of non-negative integers, which associates with every place $p \in P$ a number of tokens, $M(p)$. In the PN modeling, an event corresponds to a transition *firing*. A transition can *fire* only if the number of tokens in each input place to this transition is not less than

the weight of the arc connecting the input place to the transition. When a transition fires, the number of tokens in each input (output) place to this transition is decreased (increased) by the value of the arc weight connecting the place and the transition. The PN framework offers (i) an explicit representation of the system structure, and (ii) a compact representation of the underlying system dynamics.

In the PN modeling framework, a sequential RAS is modeled by a set of (state machine) subnets modeling the process plans of the supported process types, interconnected by a number of places modeling the resource availabilities [5, 6, 22, 64, 93]. A Petri net with such a structure is called a "process-resource net" [72]. Figure 1.3 depicts the Petri net that corresponds to the RAS given in Table 1.1. Process types J_1 and J_2 are respectively modeled by the circuits $< P_{10}, T_{10}, P_{11}, T_{11}, P_{12}, T_{12}, P_{13}, T_{13}, P_{10} >$ and $< P_{20}, T_{20}, P_{21}, T_{21}, P_{22}, T_{22}, P_{23}, T_{23}, P_{20} >$. More specifically, the three processing stages of J_1 (resp., J_2) are modeled by the three places P_{11}, P_{12}, and P_{13} (resp., P_{21}, P_{22}, and P_{23}). On the other hand, places P_{10} and P_{20} model the *"environment"*, since their tokens correspond to processes waiting their initiation and/or having finished their processing; these places are known as the process *"idle"* places, in the relevant terminology. The availability of the three resource types R_1, R_2, and R_3 is modeled by the corresponding places in the figure. Finally, the resource allocation requests posed by the various processing stages, and the complete logic of the resource allocation protocol that was described during the introduction of this example, are modeled by the connectivity of the places R_1, R_2, and R_3 to the two subnets modeling the process types J_1 and J_2.

A Petri net is *"live"iff* each of its transitions can eventually fire from each of its reachable states, i.e., given a reachable state and a transition of a live PN, there exists a sequence of feasible transitions starting from the given state that results in a state at which the given transition can fire. In the PN literature, the establishment of RAS non-blocking behavior is known as the problem of Liveness-Enforcement Supervision

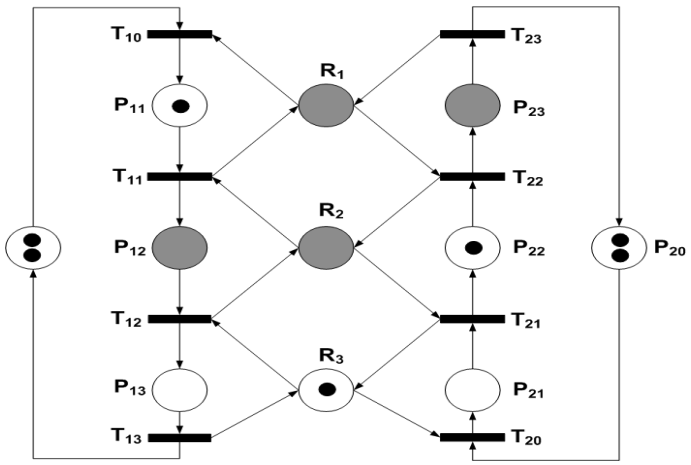


Figure 1.3: A Petri net modeling the RAS of Table 1.1. The depicted net marking corresponds to RAS state q^{17} in the FSA-based representation of the RAS dynamics of Figure 1.2. This state corresponds to a deadlock, and the shaded places highlight the empty siphon that interprets the developed deadlock.

(LES) of the corresponding process-resource net. Of particular significance in the PN-based LES methods is the *siphon* concept. Basically, a *siphon* is a set of places such that any input transition to this set is also an output transition of the set. This structural property of siphons further implies that any input transition of a siphon can be potentially disabled by some of its own places. Of particular interest are those net markings where every input transition of a siphon is experiencing this situation; siphons marked in this way are said to be "deadly marked" [72, 65], since their input transitions are permanently dead, and they can not receive any further tokens during the further evolution of the net dynamics. We also notice that, in ordinary PNs, i.e., in PNs where all the arc weights are unitary, deadly marked siphons are empty siphons. The particular net marking depicted in Figure 1.3 corresponds to the deadlock state q^{17} depicted in the FSA of Figure 1.2. In the dynamics of the net depicted in Figure 1.3, the deadlock developed between the two running process instances represented by the tokens in places P_{11} and P_{22} can be "explained" through the emptiness of places $R_1,\ R_2,\ P_{12}$, and P_{23}, that are shaded in the figure [22, 72]. It can be easily checked that this set of places constitutes an empty siphon, as defined

above.

1.4 Literature survey

As mentioned in the introductory section, the problem of deadlock avoidance has been formally abstracted since the early 60's in the context of computer systems engineering, and more recently, it has also been addressed by a broad range of other engineering communities, including manufacturing, transportation, and internet-based workflow management systems. As a result, the problem has been studied through a broad host of methodologies, ranging in their level of rigor and sophistication from the very ad-hoc to the very formal, and with varying degrees of analysis and synthesis capabilities. This entire endeavor has converged more recently, to a large extent, to the modeling abstractions and the analytical capabilities of qualitative DES theory [9], and their specialization to the considered problem context [68, 9, 88].

Another significant body of results that are currently available in the relevant literature concerns the computational complexity of the deadlock avoidance problem and the computation of the maximally permissive DAP. Along these lines, it has been established that computing the maximally permissive DAP is NP-Hard for the majority of RAS behavior [3, 29, 41].

The research community has tried to circumvent the limitations arising from these negative results by pursuing the following directions:

1. The identification of "special structures" that allow the deployment of the maximally permissive DAP through algorithms of polynomial complexity with respect to the compact representation of the underlying RAS. Some typical results can be found in [24, 93, 41, 3, 75].

2. The development of sub-optimal – i.e., non-maximally permissive – solutions which are based on polynomially assessed properties of the underlying RAS states that act as "surrogate" characterization to safety. Under these policies,

the tentative RAS transitions are allowed only if the resultant state satisfies the (policy-defining) polynomially computable property. Hence, if the reachable state subspace satisfying this property constitutes a strongly connected component containing state $\mathbf{s}_0$, in the FSA representing the underlying RAS, the system behavior will remain deadlock-free. Specific policies implementing this general idea can be found in [5, 74, 40, 39, 23, 34, 65].

3. The adoption of alternative, more compact, representations of the considered RAS dynamics in the hope that the compactness of these alternative representations, combined with further structural properties and insights revealed by them, will also lead, at least in most practical cases, to fairly compact characterizations of the target policy and to more efficient approaches for its derivation. A modeling framework that seems to hold particular promise along this line of research, and therefore, has been explored more persistently in the past, is that of Petri nets (PN) [54]. In particular, the attribution of the non-liveness of the RAS-modeling PNs to the formation of the structural objects of empty and deadly marked siphons, has led to the development of a multitude of efforts that seek to characterize the maximally permissive deadlock avoidance policy (DAP) by imposing the minimum possible amount of control that will prevent the formation of such problematic structures. The complete characterization of the RAS non-liveness through deadly marked siphons was first addressed in [65], while similar results have appeared in [79]. This finding subsequently enabled the liveness-enforcing supervision of process-resource nets through SBPI methods (i.e., supervision based on place invariants). SBPI was introduced in [27, 53, 36], and it deals with the enforcement of a set of linear inequalities on the net marking through the superimposition of some additional (control) structure upon the "plant" PN that takes the form of "monitor" places. In the case of RAS

liveness enforcing supervision, the imposed monitor places and the corresponding inequalities prevent a siphon from becoming deadly-marked. Furthermore, the introduced monitor places can be interpreted as "fictitious" resources, and therefore, the expanded net remains in the class of process-resource nets. This effect has led to the introduction of iterative procedures that, at each iteration, seek to detect a problematic siphon, and then, control it through the addition of an appropriate set of monitor places. However, the number of siphons for a given Petri net might be exponential. This problem can be partially remedied by the introduction of the concept of "elementary" siphons [45, 46, 43, 73] whose number is bounded by the number of places and transitions. The main advantage of elementary siphons is that other siphons can be controlled if the elementary ones are controlled [22, 47, 44]. In this way, the structure of the derived DAP is simplified. But a systematic complete methodology that will provide a highly permissive DAP, while taking advantage of an appropriate set of elementary siphons, is currently missing. Another major drawback of the PN-based methods is that they can be stalled, or return eventually a suboptimal solution because the maximally permissive DAP might not be expressible by a set of linear inequalities on the net marking, and therefore, it might not admit a PN-based representation. Concluding the discussion on the role of the siphon-based liveness characterization of process-resource nets, we notice that it has also enabled the development of liveness assessment methods in the form of Mixed Integer Programs that assess the liveness of a given RAS based on the structure of the corresponding PN model, without the explicit enumeration of the underlying state space [11, 65, 37].

4. Another prominent approach that has been developed primarily in the context of PN modeling but essentially spans, both, the FSA and the PN-based representations discussed earlier, is that of the "theory of regions" [4] and its

derivatives. The key problem addressed by the theory of regions is the conversion of a system modeled originally as an FSA into a Petri net such that each distinct event is represented by a single transition, and the reachability graph of the synthesized Petri net is isomorphic to the original FSA. In [81], it was proposed to use the theory of regions to design an optimal PN modeled DAP by first computing the optimal DAP using enumerative FSA-based approaches (in particular, the standard R&W SC representations and methods mentioned in the introductory section), and subsequently, encoding this policy to a PN model through the theory of regions. The approaches in [26, 81] can find an optimal supervisor if such a supervisor exists. But these approaches are also limited by the aforementioned potential inability to express the maximally permissive DAP as a PN. Furthermore, even in their feasible cases, practical experience has shown that these methods are very demanding from a computational standpoint, and they result in PN representations of the maximally permissive DAP that are much larger than the PN modeling the original RAS.

5. Another recent approach, that is reminiscent of the theory of regions, adopts an adequately compact representation for the underlying RAS in the form of an Extended Finite State Automaton (EFSA). An EFSA, as presented in [10], is an extension of the ordinary FSA that associates to each transition, a guard (conditional) formula and action functions, including different variables. In this kind of automaton, a transition is enabled *iff* the associated guard is satisfied. Furthermore, when a transition fires, the associated action functions are applied, updating the corresponding variables. In [50, 52, 51], the sought DAP is synthesized in the FSA modeling framework using Binary Decision Diagrams (BDDs) in an effort to alleviate the relevant computational effort, and then it is encoded as guarding conditions in the EFSA. However, the scalability and the size of the thus synthesized DAP are problems that yet need to be addressed

for the efficient implementation of this approach.

More extensive and comprehensive treatments of many of the results and methodologies that were outlined in the previous paragraphs, can be found in [95, 72, 69, 42]. We conclude our discussion on the key concepts and elements that underlie the current theory on the problem of deadlock avoidance in sequential RAS and the corresponding literature, by noting that, to be able to use the R&W theory, the FSA representing the underlying RAS should be computed. Several software tools are available to support the relevant computations: DESUMA, SUPREMICA, TCT, etc. [77, 2, 25]. Yet, it is well-recognized in the field that the enumeration and the representation of the underlying state space suffer from scalability issues even when symbolic techniques are employed to encode the automaton transition function [80, 28]. In recent years, some attention has been paid to this issue. For instance, the state space representation based on BDDs [8, 12, 49, 66], the data decision diagrams [18], the hierarchical set decision diagrams [19, 78], the stubborn sets [84], and the sleep-set methods for reduced state space generation [86] are some examples of this endeavor. Nevertheless, it can be safely argued that the deployment of a pertinent and adequate FSA-based representation of a DES dynamics remains a challenging task, and it can always benefit from the special structure that might be present in the considered class of DES.

CHAPTER II

THESIS OVERVIEW

2.1 A novel perspective for the RAS deadlock avoidance problem

It can be inferred from the previous chapter that the two most prominent challenges in deploying the maximally permissive DAP for any given RAS configuration are (a) the NP-hardness of the computation of the target policy, and (b) the inability of the PN modeling framework to guarantee an effective representation of the maximally permissive DAP for any given RAS configuration. We shall refer to the second limitation by saying that the corresponding framework is "*incomplete*" with respect to the maximally permissive DAP. In this work we seek to address the two limitations stated above. The first limitation is mitigated by discriminating between (i) the computational complexity that concerns the off-line computational effort necessary for acquiring the target policy, and (ii) the computational complexity that concerns the on-line implementation of this policy. In general, the computational budget for the former task might tolerate expensive computations while that for the latter will be quite stringent. On the other hand, the second limitation can be addressed by selecting an alternative policy representation that guarantees the effective representation of the maximally permissive DAP for any RAS that belongs to the RAS class defined in Chapter 1, and still retains the compactness of the PN model.

As already mentioned, the FSA-based representation of the maximally permissive DAP and its computation using the R&W SC framework is straightforward [9]. However, the size of the FSA modeling the considered RAS grows exponentially with respect to some of its more natural and more compact representations. The severity

of this problem can be mitigated if the FSA model is employed for the computation of the maximally permissive DAP only in an off-line stage, where the computational requirements are more affordable. Hence, we propose to organize the overall computation of the optimal DAP into two stages, with the first stage obtaining this policy in the R&W SC framework, and the second stage trying to express/"translate" the obtained result in a more compact form. However, instead of relying this compression to concepts and results coming from the PN modeling framework, we perceive the optimal DAP as a "dichotomy" of the RAS state space, and we essentially seek a parsimonious classifier that will effect this dichotomy. In this way, we are able to tap upon concepts, insights, and results that are coming directly from the relevant classification theory. Indeed, as it is revealed in the rest of this document, the methods pursued in this work open new ways for thinking about the considered problem that effectively complement all the previously used approaches. This new line of reasoning subsequently results into new fundamental insights, and connects the overall analysis to very classical and yet very powerful representation frameworks and techniques. The adopted approach selects a particular representation for the sought classifier that is able to provide computationally efficient classification for the RAS class under consideration. More specifically, we shall seek parametric and non-parametric representations for the classifier. Roughly speaking, in our work, a parametric classifier is defined by a set of linear inequalities and/or Boolean functions, whereas a non-parametric classifier is defined by a pertinent data structure that stores the information needed for the classification. For parametric classifiers, the classifier design problem is defined as a minimization problem over a certain parameter space that results from the adopted representation. The treatment of the classifier design problem as an explicit optimization problem also enables the development of heuristics that can effectively balance the structural optimality of the sought classifier and the computational complexity involved in its development, and of analytical bounds that

characterize the potential sub-optimality incurred by the use of these heuristics. On the other hand, for the non-parametric classifier, the problem is defined as the compression of the stored information in a way that (i) makes the classifier adequately compact, and (ii) facilitates the on-line processing of this information.

The computational tractability of the posed minimization/compression problem is facilitated by additional properties of the considered dichotomy that enable an effective compression/reduction of the information to be considered explicitly during the classifier construction process, in terms of the size and the dimensionality of the involved data sets. From a more practical standpoint, the adopted methodology has allowed the efficient implementation of the maximally permissive DAP for very large-scale RAS, with sizes and underlying state spaces way beyond of those addressed in the current literature.

2.2 Detailed problem statement

In view of the discussion that is provided in the previous section and in Chapter 1, the basic problem addressed in this thesis can be succinctly described as follows. We want to develop a methodology that will enable the effective deployment of maximally permissive deadlock avoidance in the context of the emerging technological applications that exhibit high levels of resource sharing and operational flexibility. In particular, the proposed approach will rely on the classification concept that was introduced in the previous section, and it must exhibit the following properties:

- The adopted policy representation must guarantee the effective representation of the maximally permissive DAP for every instance of the RAS class under consideration.

- The policy synthesis procedure must be computationally tractable for the RAS instances that arise in contemporary applications.

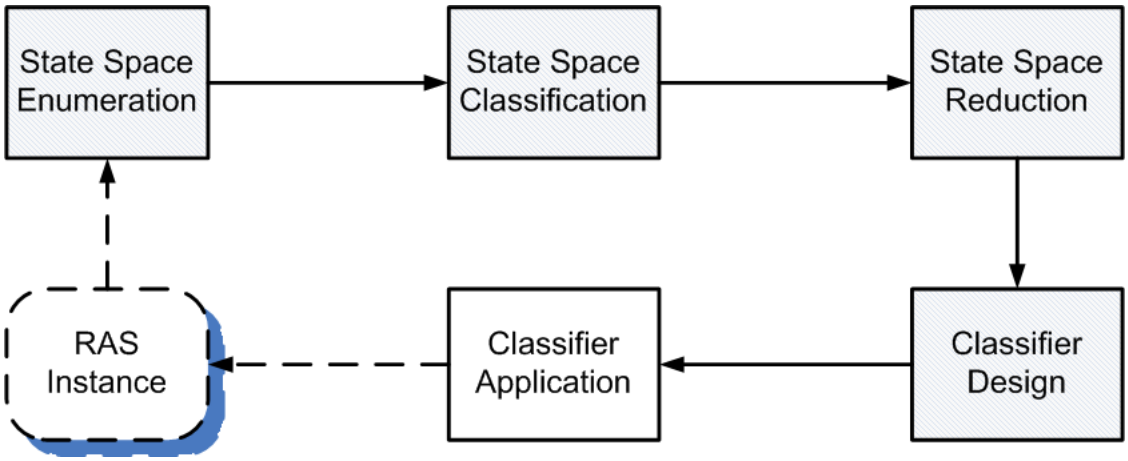


Figure 2.1: The workflow of the adopted solution for the design and the deployment of the optimal DAP. The gray stages are performed off-line, whereas the white stage is applied on-line. The dotted box refers to the considered RAS instance.

- The effective implementation of the policy must introduce the minimal possible overhead to the underlying system.

2.3 The architecture of the adopted solution

As mentioned earlier, we discriminate between the off-line computation of the optimal DAP and the on-line implementation of this policy. Figure 2.1 summarizes the workflow of the adopted solution. The first four stages are preformed off-line, whereas the last stage is applied on-line. In other words, the first four stages concern the design of the optimal DAP, whereas the last stage concerns the application of the obtained policy to the underlying RAS behavior. Next, we discuss the role of each stage in the adopted solution.

1. **The enumeration of the reachable state space S_r:** The first step in the adopted solution is to compute the FSA modeling the given RAS instance. It is well-recognized that the size of the FSA grows exponentially with respect to the compact representation of the underlying RAS. Thus, this enumeration suffers from scalability issues. To address this problem, we propose an algorithm for

the efficient enumeration of the reachable state space. The proposed algorithm is briefly discussed in Section 2.4, whereas a more detailed exposition is given in Appendix B. Furthermore, in Chapter 6, we propose an algorithm for the efficient enumeration of a critical subset of states for resolving the safety of the underlying RAS state space, while foregoing the complete enumeration of the state space.

2. **The classification of the reachable state space:** Once the reachable state space is computed, we need to classify these states into safe states (S_{rs}) and unsafe states ($S_{r\bar{s}}$). The algorithms that support this step are straightforward and well-established in the relevant literature [9, 72].

3. **The reduction/"thinning" of the classified subspaces:** A major difficulty for the systematic construction of the sought classifiers for any practical instantiation of a RAS, is the huge cardinality of the sets S_{rs} and $S_{r\bar{s}}$. This problem can be effectively addressed as follows: First, the sought classifier needs to discriminate successfully only between the set S_{rs} and the subset of the set $S_{r\bar{s}}$ that contains all the states $\mathbf{s} \in S_{r\bar{s}}$ that are reachable from some state $\mathbf{s}' \in S_{rs}$ in a single transition. We shall denote this subset of $S_{r\bar{s}}$ by $S^b_{r\bar{s}}$, and we shall refer to its elements as the *"boundary"* reachable unsafe states. Moreover, we utilize some additional *structure* presented by sets S_{rs} and $S^b_{r\bar{s}}$ in order to extract two subsets from these sets, that are of significantly reduced cardinalities compared to their respective supersets, and when used as input to the classifier design process, they result to a classifier that still classifies correctly the original sets S_{rs} and $S^b_{r\bar{s}}$. In particular, we notice that, in the state space implied by Equation 1.1, the states dominated, component-wise, by some safe state are also safe, and similarly, the states that dominate some unsafe state are also unsafe. The reduction/thinning step is designed to take advantage of this *"monotonicity"*

effect by focusing the classifier design on the maximal elements of S_{rs} and the minimal elements of $S^b_{r\bar{s}}$. The detailed description of the reduction stage is the content of Section 2.5.

4. **The classifier design:** This stage constitutes the core of the DAP design problem. The most important issue in this stage concerns the selection of the particular *structure* that will be employed for the sought classifier. For the case of parametric classifiers, the simplest classification structure is that provided by a system of linear inequalities; such a classifier will be referred to as a "*linear classifier*" in the sequel. Besides its structural simplicity, a linear classifier is also easier to understand and analyze from the standpoint of structural minimality, and it is also compatible with additional representations of the maximally permissive DAP that have been pursued in the past. More specifically, as remarked in Section 1.4, a representation of the maximally permissive DAP as a set of linear inequalities renders it implementable in the PN modeling framework, and endows upon it all the analytical and computational advantages that are possessed by this framework. On the other hand, the discussion of Section 1.4 also revealed that the representation of the maximally permissive DAP as a set of linear inequalities is not always feasible in the context of the considered RAS class, and therefore, there is a need for more powerful classification structures, that will be able to provide a complete solution for the classifier-design problem arising in the considered RAS class. Motivated by the above remarks, in Chapter 3, we shall start by addressing the linear classifier design problem in the context of the Gadara RAS, a RAS class for which the maximally permissive DAP can always be characterized with a set of linear inequalities. Next, in Chapters 4– 7, we shall utilize more powerful classification structures to encompass broader RAS classes.

5. **The real-time application of the constructed DAP:** Once the classifier encoding the optimal DAP is synthesized through the previous stages, we can implement this policy by allowing a tentative event to fire *iff* the resultant state is classified as a safe state by the synthesized classifier.

With the overall architecture of the adopted solution defined above, the rest of the chapter is organized as follows: In Section 2.4, an efficient algorithm that supports Stage 1 is described. In Section 2.5, a detailed description of the Stage 3 is provided. Finally, Section 2.6 provides a breakdown of the remaining tasks for the complete development of the proposed approaches, and thus, it defines the organization for the rest of this thesis.

2.4 An efficient enumeration of the reachable state space

In this section, we overview an algorithm for the generation and storage of the reachable space, S_r, that has been found to be especially efficient in our computational studies. A detailed description and analysis of the algorithm can be found in Appendix B. This algorithm provides an enumeration of S_r, by first identifying, as an intermediary step, all the states corresponding to a feasible resource allocation, according to the prevailing resource capacity constraints (c.f., Eq. 1.1); we shall refer to these RAS states as *"valid"* states, and the corresponding state set will be denoted by S_v. Once S_v has been constructed, a subsequent procedure filters out from it, the set of reachable states S_r. Therefore, the whole computation is organized naturally into two major procedures: (a) Algorithm B.1 of Appendix B is utilized to generate the state set S_v, and (b) Algorithm B.2 of the same appendix is utilized to reduce S_v to S_r. The introduction of the intermediate step of generating the state set S_v does not increase substantially the complexity of the involved computation, since, as it is revealed in Appendix B, both sets S_v and S_r are of comparable sizes. On the other hand, performing the overall computation through the proposed sequence enables a

more efficient handling and storage of the information characterizing the underlying FSA structure, and also the effective utilization of the auxiliary memory in case that the involved data structures grow so big that they cannot be accommodated in the core memory.

2.5 The reduction/"thinning" of the classified subspaces

We start the description of this task by establishing that the subspaces S_{rs} and $S_{r\bar{s}}$ present some additional structure that will become useful in the design of the target classifiers. The next definition will facilitate the formal statement and development of the relevant results.

Definition 2.1 *Let* $\mathbf{x},\ \mathbf{x}'$ *denote two vectors in the vector space* $(\mathbb{Z}_0^+)^\xi$. *Then, the relationship "*$\preceq$*" imposes a partial ordering on* $(\mathbb{Z}_0^+)^\xi$ *that is defined by:*

$$\mathbf{x} \preceq \mathbf{x}' \iff (\forall i = 1, \ldots \xi, \quad \mathbf{x}[i] \leq \mathbf{x}'[i]) \tag{2.1}$$

Furthermore, $\mathbf{x} \prec \mathbf{x}'$ *(resp.* $\mathbf{x} \succ \mathbf{x}'$*) will denote the fact that* $\mathbf{x} \preceq \mathbf{x}'$ *(resp.* $\mathbf{x} \succeq \mathbf{x}'$*), and there is at least a pair of components* $\mathbf{x}[i]$, $\mathbf{x}'[i]$ *for which the corresponding inequality is strict.*

It should be clear that the ability of the activated processes in a given RAS state $\mathbf{s} \in S$ to proceed to completion, depends on the existence of a sequence $(\mathbf{s}^{(0)} \equiv \mathbf{s}, e^{(1)}, \mathbf{s}^{(1)}, e^{(2)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(n-1)}, e^{(n)}, \mathbf{s}^{(n)} \equiv \mathbf{s}_0)$, such that at every state $\mathbf{s}^{(i)},\ i = 0, 1, \ldots, n-1$, the free (or "slack") resource capacities at that state enable the job advancement corresponding to event $e^{(i+1)}$. Furthermore, if such a terminating sequence exists for a given state $\mathbf{s}$, then the event feasibility condition defined by Equation 1.1 implies that this sequence will also provide a terminating sequence for every other state $\mathbf{s}' \preceq \mathbf{s}$. On the other hand, if state $\mathbf{s}'$ possesses no terminating sequences, then it can be safely inferred that no such terminating sequences will exist for any other state $\mathbf{s} \succeq \mathbf{s}'$ (since, otherwise, there should also exist a terminating sequence for $\mathbf{s}'$, according to

the previous remark). The next proposition provides a formal statement to the above observations.

Proposition 2.1 *[74, 71] Let* $\mathbf{s}$, $\mathbf{s}'$ *denote two states of a given RAS* Φ. *Then,*

1. $\mathbf{s} \in S_s \ \wedge \ \mathbf{s}' \preceq \mathbf{s} \implies \mathbf{s}' \in S_s$

2. $\mathbf{s}' \in S_{\bar{s}} \ \wedge \ \mathbf{s}' \preceq \mathbf{s} \implies \mathbf{s} \in S_{\bar{s}}$

In light of Proposition 2.1, we define the concepts of *maximal reachable safe state*, *minimal reachable unsafe state*, and *minimal reachable deadlock state*, that will play an important role in the subsequent developments:

Definition 2.2 *Given a RAS* $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$,

1. *a reachable safe state* $\mathbf{s} \in S_{rs}$ *is a* maximal reachable safe state iff $\neg\exists$ *a reachable safe state* $\mathbf{s}' \in S_{rs}$ *such that* $\mathbf{s}' \succ \mathbf{s}$;

2. *a reachable unsafe state* $\mathbf{s} \in S_{r\bar{s}}$ *is a* minimal reachable unsafe state iff $\neg\exists$ *a reachable unsafe state* $\mathbf{s}' \in S_{r\bar{s}}$ *such that* $\mathbf{s}' \prec \mathbf{s}$.

3. *a reachable deadlock state* $\mathbf{s} \in S_{rd}$ *is a* minimal reachable deadlock state iff $\neg\exists$ *a reachable deadlock state* $\mathbf{s}' \in S_{rd}$ *such that* $\mathbf{s}' \prec \mathbf{s}$.

Also, in the sequel, the set of maximal reachable safe states will be denoted by $\hat{S}_{rs}$, *and the set of minimal reachable unsafe states will be denoted by* $\hat{S}_{r\bar{s}}$. *Finally, the set of minimal reachable deadlock states will be denoted by* $\hat{S}_{rd}$.

An additional implication of Proposition 2.1 that will be useful in the subsequent developments is stated in the following lemma:

Lemma 2.1 *Given a RAS* $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$, *the state set* S_{rs} *contains all possible chains of integer vectors between the origin* $\mathbf{0}$ *and its maximal elements* $\mathbf{s} \in \hat{S}_{rs}$.

Proof: First, we argue that all possible chains of integer vectors between the origin $\mathbf{0}$ and the elements of $\hat{S}_{rs}$ belong in S_r. Indeed, consider a state $\mathbf{s}$ belonging in an integer vector chain from state $\mathbf{0}(\equiv \mathbf{s}_0)$ to an element $\mathbf{s}' \in \hat{S}_{rs}$. Since $\mathbf{s}' \in \hat{S}_{rs}$, it is a reachable state, and therefore, there exists an event sequence σ' such that $\mathbf{s}' = \tilde{f}(\mathbf{s}_0, \sigma')$. Furthermore, from the specification of the states $\mathbf{s}$ and $\mathbf{s}'$ it also holds that $\mathbf{s} \preceq \mathbf{s}'$, and therefore, the process instances contained in $\mathbf{s}$ is a subset of the process instances contained in $\mathbf{s}'$. But since, as implied by the RAS definition, the various process instances do not interact with each other except for the sharing of the system resources, it is easy to see that the reachability of state $\mathbf{s}'$ implies also the reachability of state $\mathbf{s}$; a corresponding event sequence σ can be obtained from the event sequence σ' mentioned above by removing from it all the events concerning process instances not belonging in $\mathbf{s}$.

The fact that the considered state $\mathbf{s}$ belongs also in S_s, and therefore, in S_{rs}, results from Proposition 2.1 and the aforestated fact that $\exists \mathbf{s}' \in \hat{S}_{rs}$ s.t. $\mathbf{s} \preceq \mathbf{s}'$. $\square$

As previously mentioned, the goal of the reduction stage is to extract two subsets of the sets S_{rs} and $S_{r\bar{s}}$ that are of significantly reduced cardinalities compared to their respective supersets, and when used as input to the classifier design process, they result to a classifier that encodes the sought dichotomy, and hence, can be employed for the deployment of the optimal DAP. This is done through the following successive thinning operations:

1. **Thinning the set $S_{r\bar{s}}$ by focusing on its boundary to the reachable and safe subspace**: We have already pointed out that the effective implementation of the maximally permissive DAP for a given RAS is equivalent to the recognition and the blockage of transitions from the safe to the unsafe region of the underlying state space S. This implies that the classifier design process can focus only on the sets S_{rs} and $S^b_{r\bar{s}}$.

2. **Thinning the sets S_{rs} and $S^b_{r\bar{s}}$ by respectively focusing on their maximal and minimal elements**: We shall show that for all the classifier representations proposed in this work, it is possible to obtain a classifier for the entire sets S_{rs} and $S^b_{r\bar{s}}$, by focusing the classifier design process only on the maximal elements of the first set, $\hat{S}_{rs}$, and the minimal elements of the second set, $\hat{S}^b_{r\bar{s}}$.

3. **Converting the separation problem of $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ to an equivalent separation problem of reduced dimensionality**: In our numerical experimentation, we have frequently encountered a situation where many components of the vectors included in the set $\hat{S}_{rs}$ are always greater than or equal to the corresponding components of the vectors included in the set $\hat{S}^b_{r\bar{s}}$. The removal of these components from further consideration, through the orthogonal projection of the vector sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ to the subspace defined by the remaining components, retains all the information that is necessary for the development of a classifier that will separate correctly the original state subsets S_{rs} and $S^b_{r\bar{s}}$. To formalize further the ideas expressed in the above discussion, let V denote the ξ-dimensional vector space supporting the vector sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$, L denote the set of dimensions of space V, and L_0 denote the set of dimensions that have the property that:

$$\forall i \in L_0,\ \forall \mathbf{s} \in \hat{S}_{rs},\ \forall \mathbf{u} \in \hat{S}^b_{r\bar{s}} :\ \mathbf{s}[i] \geq \mathbf{u}[i] \tag{2.2}$$

This set of dimensions is removed by the proposed projection P. Let $L_P \equiv L \backslash L_0$, and V_P denote the $|L_P|$-dimensional subspace supporting the projection P. Also, let $\Pi\colon \mathbb{N} \rightarrow \mathbb{N}$ be a bijection that maps the elements of the dimension set L_P to the dimensions of subspace V_P. Finally, in the following, we shall let $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$ denote respectively the images of the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ under P.

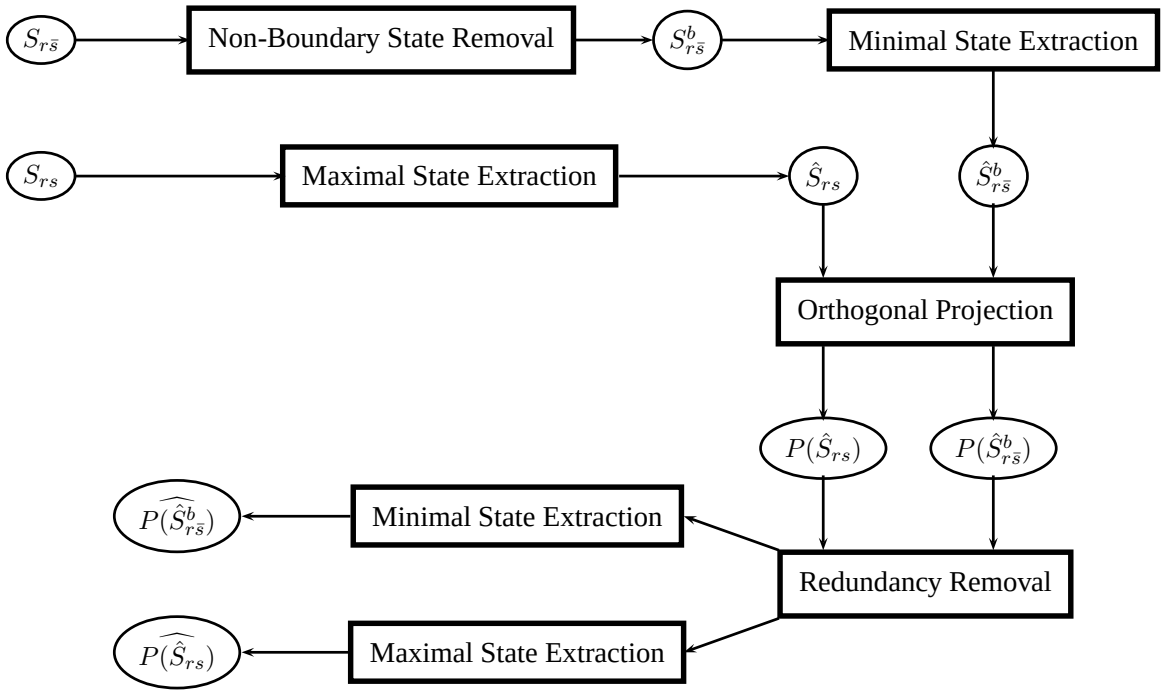


Figure 2.2: The steps of the adopted state space reduction/thinning process.

4. **Some further simplifications**: The projection P introduced in the previous item is not bijective, and therefore, it may introduce some redundancy among the elements of $P(\hat{S}_{rs})$ and the elements of $P(\hat{S}^b_{r\bar{s}})$. Also the removal of the components in L_0 can introduce some dominance among the elements of both sets with respect to the ordering '$\preceq$'. Hence, the redundant and the non-maximal (resp., non-minimal) elements should be removed from the set $P(\hat{S}_{rs})$ (resp., $P(\hat{S}^b_{r\bar{s}})$). The resulting sets will be denoted by $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$.

These thinning operations impose some restrictions on the classifier design, leading in some cases to an increase of the size of the attained classifier. On the other hand, the proposed thinning operations have allowed the efficient implementation of the maximally permissive DAP for very large-scale RAS, with sizes and underlying state spaces way beyond of those addressed in the past literature.

Figure 2.2 summarizes the data thinning process described in the previous paragraphs. Table 2.1 demonstrates the results that are obtained by the application of this thinning process to the sets S_{rs} and $S_{r\bar{s}}$ corresponding to the safe and unsafe

Table 2.1: The various sets obtained by the application of the data thinning process of Figure 2.2 to the sets S_{rs} and $S_{r\bar{s}}$ corresponding to the reachable safe and unsafe subspaces of the RAS of Table 1.1.

$S^b_{r\bar{s}} = S_{r\bar{s}}$
$\hat{S}_{rs} = \{q^{13} \equiv [1\ 1\ 1\ 0\ 0\ 0]^T,\ q^{14} \equiv [0\ 0\ 0\ 1\ 1\ 1]^T\}$ $\hat{S}^b_{r\bar{s}} = \{q^{15} \equiv [1\ 0\ 0\ 1\ 0\ 0]^T,\ q^{16} \equiv [0\ 1\ 0\ 1\ 0\ 0]^T,$ $q^{17} \equiv [1\ 0\ 0\ 0\ 1\ 0]^T\}$ $\hat{S}_{rd} = \{q^{16} \equiv [0\ 1\ 0\ 1\ 0\ 0]^T,\ q^{17} \equiv [1\ 0\ 0\ 0\ 1\ 0]^T\}$
$L_0 = \{3,\ 6\}$ $\widehat{P(\hat{S}^b_{r\bar{s}})} = P(\hat{S}^b_{r\bar{s}}) = \{[1\ 0\ 1\ 0]^T,\ [0\ 1\ 1\ 0]^T,\ [1\ 0\ 0\ 1]^T\}$ $\widehat{P(\hat{S}_{rs})} = P(\hat{S}_{rs}) = \{[1\ 1\ 0\ 0]^T,\ [0\ 0\ 1\ 1]^T\}$

subspaces of the RAS of Table 1.1.

2.6 Task breakdown

Given the solution architecture introduced above and the thesis objectives stated in Section 2.2, the remaining tasks addressed by this thesis are broken down as follows:

1. First, in Chapter 3, we address the problem of the synthesis of the parsimonious classifiers representing the maximally permissive DAP in the context of the RAS sub-class with resource types of unit capacity, i.e., in the Gadara RAS. We establish that, for any Gadara RAS instance, the sought separation can be effected by a linear classifier. We also provide efficient computational procedures that can support the synthesis of the target classifiers for RAS instances with very large state spaces.

2. The aforestated approach is extended to handle any RAS from the broader class defined in Chapter 1. The elimination of the resource capacity constraint that defines the Gadara RAS, complicates the structure of the RAS reachable subspace, leading to the inability of the linear classifiers to effect the sought

separation. This effect necessitates the shift of the classifier representation to a more general/powerful structure. We propose two representations for the sought classifier, a parametric representation and a non-parametric representation. We establish their completeness with respect to the considered classification problem, and we also provide the design and optimization methods to synthesize the target classifiers. The parametric representation is given in Chapter 4, whereas the non-parametric representation is given in Chapter 5.

3. In order to address the potential limitations that can result from the high computational complexity involved in the enumeration of the FSA modeling the RAS behavior, during the first stage of our approach, we propose a novel algorithm that can support the enumeration of all the minimal reachable unsafe states ($\hat{S}_{r\bar{s}}$), while foregoing the complete enumeration of the state space of the underlying FSA. Chapter 6 gives the details of this algorithm.

4. Finally, in Chapter 7, we address the design of the maximally permissive DAP for R/W-RAS. A main challenge for the R/W-RAS is that the resource capacity of a resource type that models a R/W lock, is infinite in the reading mode. As a result of this new effect, the state automaton modeling the underlying R/W-RAS behavior is not necessarily finite. However, as it will be proved in Chapter 7, the set of minimal reachable unsafe states ($\hat{S}_{r\bar{s}}$) of the R/W-RAS is finite. Its complete enumeration is performed by an adaptation of the algorithm developed in Task 3.

Concluding this chapter, we would like to remark that the results obtained from these four tasks were reported in [59, 62, 58, 55, 57, 61, 56, 60].

CHAPTER III

LINEAR CLASSIFIERS

3.1 Introduction

As explained in the previous chapter, the proposed solution distinguishes between the off-line and the on-line computation that is required for the effective implementation of the maximally permissive DAP, and it seeks to develop representations of this policy that will require minimal on-line computation. The particular representation that we adopt in this chapter is that of a parsimonious linear classifier that will effect the underlying dichotomy of the reachable state space into safe and unsafe subspaces. As indicated in the previous chapters, linear classifiers are not always capable of establishing the sought separation for the RAS classes defined in Chapter 1. On the other hand, as it will be shown in the subsequent developments, any instance of Gadara RAS accepts the separation of the underlying safe and unsafe subspaces through a set of linear inequalities. Hence, in this chapter, we restrict our attention to the Gadara RAS. We remark that the linear classification design methodology developed in this chapter can also be applied to broader RAS classes, but in that case, there is no guarantee of its completeness[1].

In light of the above positioning, the rest of the chapter is organized as follows: Section 3.2 discusses the properties of the Gadara RAS and the implication of these

[1]Another RAS subclass for which the maximally permissive DAP is always representable as a linear classifier, is the subclass of Disjunctive, Single-Unit RAS where the set of unsafe states coincides with the set of deadlock states. In particular, it can be seen that, in the case of Single-Unit RAS, the prevention of any (reachable) deadlock can be achieved in a maximally permissive manner through the enforcement of a linear inequality stipulating that the total number of process instances executing the processing stages that are involved in the considered deadlock remains strictly below the combined capacity of the resources that are involved in the deadlock. A detailed characterization of the aforementioned RAS class can be found in [41, 72].

properties on the classifier design problem. Section 3.3, first, provides a formal definition of the classification problem to be considered in this chapter, and subsequently, it proceeds to its reduction to an equivalent classification problem with a much smaller input set in terms of the explicitly considered state vectors and their dimensionality, along the lines discussed in Section 2.5. Section 3.4 addresses the synthesis of a linear classifier for the reduced classification problem, by formulating and solving this problem as a mixed integer program. On the other hand, Section 3.5 introduces a heuristic approach to the synthesis of the sought classifiers. Section 3.6 reports a series of computational experiments that demonstrates the feasibility of the methodological approaches proposed in the chapter, and reveals the extensive computational gains that are obtained by them. Finally, Section 3.7 concludes the chapter.

3.2 The binary nature of the state space of the Gadara RAS and its implications

We remind the reader that a Gadara RAS Φ_g is characterized by a single unit capacity for all the resource types. The single unit capacity assumption, in conjunction with Equation 1.1 and the non-zero nature of the resource allocation requests $\mathcal{A}_{jk}$, imply that the RAS state vector $\mathbf{s}$ is of a binary nature; i.e., the state space S of the DFSA $G(\Phi_g)$, as well as any other subspace of S, consists of a number of extreme points on the ξ-dimensional hypercube $\mathcal{C}$ defined by

$$\mathcal{C} \equiv \{(x_1, x_2, \ldots, x_\xi) : \ 0 \leq x_i \leq 1, \ \ \forall i = 1, \ldots, \xi\} \tag{3.1}$$

Next we show that every extreme point of $\mathcal{C}$ can be effectively separated from the rest by a single linear inequality.

Proposition 3.1 *Consider the hypercube $\mathcal{C}$ defined by Equation 3.1, and let $\mathbf{x} = (x_1, x_2, \ldots, x_\xi)$ denote one of its extreme points. Then, point $\mathbf{x}$ can be separated from*

the remaining extreme points of $\mathcal{C}$ *by the linear inequality* $\mathbf{a}^T\mathbf{x} \leq b$ *where*

$$\mathbf{a}[i] := \begin{cases} 1, & \textit{if } \mathbf{x}[i] = 1 \\ -1, & \textit{if } \mathbf{x}[i] = 0 \end{cases} \quad ; \quad b := \sum_{i=1}^{\xi} \mathbf{x}[i] - 1 \tag{3.2}$$

Proof: Under the assignments of Equation 3.2, $\mathbf{a}^T\mathbf{x} = \sum_{i=1}^{\xi} \mathbf{x}[i] > b$. On the other hand, the mis-alignment of the unit components of any other extreme point $\mathbf{x}'$ with the positive elements of vector $\mathbf{a}$ implies that $\mathbf{a}^T\mathbf{x}' \leq b$. □

The practical implications of Proposition 3.1 for the DAP design problem that are addressed in the rest of this chapter, are stated in the following corollary:

Corollary 3.1 *Consider two sets* X *and* $\tilde{X}$ *that consist of binary vectors from some* ξ*-dimensional space, and further assume that* $\tilde{X} \subset X$*. Then, there exists a system of linear inequalities* $\{\mathbf{A}[i,.] \cdot \mathbf{x} \leq \mathbf{b}[i],\ i = 1, \ldots, \nu\}$ *that is satisfied by every* $\mathbf{x} \in \tilde{X}$ *and it is violated by every* $\mathbf{x} \in X \backslash \tilde{X}$*, and therefore, it can function as a linear classifier for* $\tilde{X}$ *and* $X \backslash \tilde{X}$*.*

Proof: Consider the separating inequalities that are implied by Proposition 3.1 for each $\mathbf{x} \in X \backslash \tilde{X}$. Then, the system of linear inequalities that is defined by the conjunction of all these inequalities is satisfied by every vector $\mathbf{x} \in \tilde{X}$. At the same time, its construction implies that it is violated by any vector $\mathbf{x} \in X \backslash \tilde{X}$. □

3.3 The linear classifier design problem in the context of the Gadara RAS and its simplification

The classification problem considered in this chapter This section considers the problem of synthesizing a parsimonious classifier that will separate the reachable safe subspace S_{rs} from the reachable unsafe subspace $S_{r\bar{s}}$, for any given RAS Φ_g that belongs to the class of Gadara RAS. Corollary 3.1 of the previous section guarantees the existence of a set of linear inequalities that will perform the requested separation. Our objective is to find the minimum number of linear inequalities that achieves the

separation. The next definition provides a formal characterization of the concepts that are necessary for the exact positioning of our problem:

Definition 3.1 *Consider two vector sets G and H from an ξ-dimensional vector space V.*

1. *We shall say that sets G and H are* linearly separated *by a set of k linear inequalities $\{(\mathbf{A}[i,.], \mathbf{b}[i]) : i \in \{1, \ldots, k\}\}$* iff

$$\begin{aligned} \forall \mathbf{g} \in G: \ \forall i \in \{1, \ldots, k\}, \ \mathbf{A}[i,.] \cdot \mathbf{g} \leq \mathbf{b}[i] \quad \wedge \\ \forall \mathbf{h} \in H : \exists i^* \in \{1, \ldots, k\}, \ \mathbf{A}[i^*,.] \cdot \mathbf{h} > \mathbf{b}[i^*] \end{aligned} \tag{3.3}$$

2. *A linear classifier – or separator – for vector sets G and H is* minimal, iff *it uses the minimum possible number of linear inequalities that can separate these two sets.*

Hence, the problem addressed in this chapter can be succinctly stated as follows:

Definition 3.2 – The linear classification problem *Given a Gadara RAS Φ_g, construct a minimal linear separator for the vector sets corresponding to the sub-spaces S_{rs} and $S_{r\bar{s}}$, i.e., the reachable safe and the reachable unsafe states of the considered RAS Φ_g.*

We would like to highlight that Definitions 3.1-3.2 imply some asymmetry in the separation among S_{rs} and $S_{r\bar{s}}$. While the approach pursued in this chapter can allow for symmetric separation roles, the adopted asymmetry is necessary to enable the translation of the linear classifier into the Petri net formalism using SBPI [27, 53]. Thus, the synthesized linear classifier can be encoded into the Petri net that represents the underlying RAS, resulting in a live Petri net. A main advantage of the Petri net formalism is that the control logic can be implemented in a distributed (non-centralized) manner. While the centralized control requires to know the RAS state

before firing each event, the distributed control requires to monitor only specific groups of processing stages where the interaction among the stages of each group might introduce a deadlock. Hence, when compared to the centralized control, the distributed control allows for more concurrency in the behavior of the underlying RAS. This advantage is the main motivation for the asymmetry adopted in Definitions 3.1-3.2.

A major difficulty for the systematic construction of the aforementioned classifier for any practical instantiation of the Gadara RAS, is the huge cardinality of the sets S_{rs} and $S_{r\bar{s}}$. To circumvent this obstacle, we utilize the thinning techniques introduced in Section 2.5. To be able to use these thinning techniques, we need to restrict the coefficients of the linear inequalities to be non-negative. Hence, in the following, we show that: (i) under the aforementioned restriction, the thinning techniques of Section 2.5 are valid with respect to (w.r.t.) the linear classification problem, and (ii) the sign restriction for the classifier coefficients does not compromise either the feasibility or the optimality of the synthesized classifiers.

Thinning the sets S_{rs} and $S^b_{r\bar{s}}$ by respectively focusing on their maximal and minimal elements The next three propositions establish that it is possible to obtain a minimal linear classifier for the entire sets S_{rs} and $S^b_{r\bar{s}}$, by focusing the classifier design process only on the maximal elements of the first set, $\hat{S}_{rs}$, and the minimal elements of the second set, $\hat{S}^b_{r\bar{s}}$.

Proposition 3.2 *Any linear separator* $(\mathbf{A}, \mathbf{b})$ *for the sets* $\hat{S}_{rs}$ *and* $\hat{S}^b_{r\bar{s}}$ *with non-negative coefficients* $\mathbf{A}[i,j]$ *and* $\mathbf{b}[i]$*, is also an effective separator for the entire sets* S_{rs} *and* $S^b_{r\bar{s}}$*.*

Proof: Let $\mathbf{s} \in S_{rs}$ be an arbitrary non-maximal reachable safe state vector, and $\mathbf{s}^* \in \hat{S}_{rs}$ be a maximal reachable safe state vector such that $\mathbf{s}^* \succ \mathbf{s}$.

Also, let $\mathbf{u} \in S^b_{r\bar{s}}$ be an arbitrary non-minimal boundary reachable unsafe state

vector, and $\mathbf{u}^* \in \hat{S}^b_{r\bar{s}}$ be a minimal boundary reachable unsafe state vector such that $\mathbf{u}^* \prec \mathbf{u}$.

Finally, let $\mathbf{A}[i,.]$ be the vector of coefficients for the i-th hyperplane separating the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$, $i = 1, \ldots, k$. Then, according to the stated assumptions, $\mathbf{A}[i,.] \geq \mathbf{0}$, $\forall i \in \{1, \ldots, k\}$. Also, according to the definition of linear separation: $\forall \mathbf{s}' \in \hat{S}_{rs}$: $\forall i \in \{1, \ldots, k\}$, $\mathbf{A}[i,.] \cdot \mathbf{s}' \leq \mathbf{b}[i]$ and $\forall \mathbf{u}' \in \hat{S}^b_{r\bar{s}} : \exists i_{u'} \in \{1, \ldots, k\}$ such that $\mathbf{A}[i_{u'},.] \cdot \mathbf{u}' > b[i_{u'}]$. But then, the non-negativity of $\mathbf{A}[i,.]$, combined with the presumed relations of $\mathbf{s}$ to $\mathbf{s}^*$ and of $\mathbf{u}$ to $\mathbf{u}^*$, further imply that:

- $\forall i : \ \mathbf{A}[i,.] \cdot \mathbf{s} \leq \mathbf{A}[i,.] \cdot \mathbf{s}^* \leq \mathbf{b}[i]$
- $\exists i_{u^*} : \ \mathbf{A}[i_{u^*},.] \cdot \mathbf{u} \geq \mathbf{A}[i_{u^*},.] \cdot \mathbf{u}^* > \mathbf{b}[i_{u^*}]$

Since states $\mathbf{s}$ and $\mathbf{u}$ were arbitrarily chosen, we can infer that $\forall \mathbf{s} \in S_{rs} : \ \forall i \in \{1, \ldots, k\}$, $\mathbf{A}[i,.] \cdot \mathbf{s} \leq \mathbf{b}[i]$ and $\forall \mathbf{u} \in S^b_{r\bar{s}} : \exists i_u \in \{1, \ldots, k\}$, $\mathbf{A}[i_u,.] \cdot \mathbf{u} > \mathbf{b}[i_u]$, which means that the separator $(\mathbf{A}, \mathbf{b})$ is also an effective separator for S_{rs} and $S^b_{r\bar{s}}$. □

Proposition 3.2 implies that, under the restriction of the non-negativity of the coefficients of the linear classifier, we can focus the classifier design only on the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$. The next lemma and the following proposition establish that this restriction does not compromise either the feasibility or the optimality of the classifier construction process.

Lemma 3.1 *Consider a set $\mathcal{T} \subseteq \mathbb{N}^\xi$ such that*

$$\forall \mathbf{x} \in \mathcal{T}, \forall \mathbf{x}' \in \mathbb{N}^\xi, \mathbf{x}' \preceq \mathbf{x} \implies \mathbf{x}' \in \mathcal{T} \tag{3.4}$$

Let $Conv(\mathcal{T})$ denote the convex hull of $\mathcal{T}$, and let $\mathbf{x}_1 \in Conv(\mathcal{T})$. Then, $Conv(\mathcal{T})$ also contains any other vector $\mathbf{x}_2 \in \mathbb{R}^\xi$ such that $\mathbf{0} \preceq \mathbf{x}_2 \preceq \mathbf{x}_1$.

Proof: Assume that $\mathbf{x}_2 = diag(t_1, t_2, \ldots, t_\xi)\ \mathbf{x}_1$, where diag() implies a diagonal matrix with its principal diagonal consisting of the quoted elements, and, without

loss of generality,

$$0 \leq t_1 \leq t_2 \leq \ldots \leq t_\xi \leq 1 \tag{3.5}$$

Let $\mathbf{y}_1, \ldots, \mathbf{y}_l$ be a set of vectors in $\mathcal{T}$ such that

$$\mathbf{x}_1 = \sum_{i=1}^{l} \theta_i \cdot \mathbf{y}_i;\ \sum_{i=1}^{l} \theta_i = 1;\ \theta_i > 0,\ \forall i = 1:l \tag{3.6}$$

Define the vectors $\mathbf{y}_{ij},\ i = 1:l,\ j = 1:\xi$, such that:

$$\mathbf{y}_{ij}[k] \equiv \begin{cases} 0 & \text{if } 1 \leq k \leq j \\ \mathbf{y}_i[k] & \text{if } j < k \leq \xi \end{cases} \tag{3.7}$$

Equation 3.4 implies that all vectors $\mathbf{y}_{ij}$ belong to $\mathcal{T}$. Also, note that $\mathbf{y}_{i\xi} = \mathbf{0},\ \forall i$. Define the vectors $\mathbf{z}_i \equiv diag(t_1, t_2, \ldots, t_\xi)\ \mathbf{y}_i,\ i = 1:l$. Also, for notational convenience, define $t_0 = 0$, $t_{\xi+1} = 1$, and $\mathbf{y}_{i0} = \mathbf{y}_i,\ \forall i$. We can see that:

$$\begin{aligned} \textstyle\sum_{j=1}^{\xi+1}(t_j - t_{j-1}) \cdot \mathbf{y}_{i,j-1} &= t_1 \cdot \mathbf{y}_i + (t_2 - t_1) \cdot \mathbf{y}_{i1} + (t_3 - t_2) \cdot \mathbf{y}_{i2} + \ldots + (1 - t_\xi) \cdot \mathbf{y}_{i\xi} \\ &= t_1 \cdot (\mathbf{y}_i[1], \mathbf{y}_i[2], \ldots, \mathbf{y}_i[\xi])^T + (t_2 - t_1) \cdot (0, \mathbf{y}_i[2], \ldots, \mathbf{y}_i[\xi])^T + \ldots \\ &\quad + (t_\xi - t_{\xi-1}) \cdot (0, \ldots, 0, \mathbf{y}_i[\xi])^T + (1 - t_\xi) \cdot (0, 0, \ldots, 0)^T \\ &= (t_1 \cdot \mathbf{y}_i[1], t_2 \cdot \mathbf{y}_i[2], \ldots, t_\xi \cdot \mathbf{y}_i[\xi])^T = diag(t_1, t_2, \ldots, t_\xi)\ \mathbf{y}_i = \mathbf{z}_i \end{aligned} \tag{3.8}$$

But then,

$$\begin{aligned} \sum_{i=1}^{l} \theta_i \cdot (\sum_{j=1}^{\xi+1} (t_j - t_{j-1}) \cdot \mathbf{y}_{i,j-1}) = \sum_{i=1}^{l} \theta_i \cdot \mathbf{z}_i = \\ \sum_{i=1}^{l} \theta_i \cdot (diag(t_1, t_2, \ldots, t_\xi)\ \mathbf{y}_i) = diag(t_1, t_2, \ldots, t_\xi)\ \sum_{i=1}^{l} \theta_i \cdot \mathbf{y}_i = \mathbf{x}_2 \end{aligned} \tag{3.9}$$

Furthermore, from the expansion of the telescopic series below, it can be seen that:

$$\sum_{i=1}^{l} \theta_i \cdot \sum_{j=1}^{\xi+1} (t_j - t_{j-1}) = \sum_{i=1}^{l} \theta_i \cdot 1 = 1 \tag{3.10}$$

Finally, from Equations 3.5 and 3.6, we have that

$$\theta_i \cdot (t_j - t_{j-1}) \geq 0,\ \ \forall i = 1:l,\ \forall j = 1:\xi+1 \tag{3.11}$$

But then, the fact that $\mathbf{x}_2 \in Conv(\mathcal{T})$ results from Equations 3.9–3.11 and the fact that $\mathbf{y}_{ij} \in \mathcal{T},\ \forall i = 1:l,\ \forall j = 0:\xi.\ \square$

Proposition 3.3 *For any two sets* $\mathcal{T},\ \bar{\mathcal{T}} \subseteq \mathbb{N}^{\xi}$ *such that* $\mathcal{T}$ *and* $\bar{\mathcal{T}}$ *are linearly separable, if it also holds that*

$$\forall \mathbf{x} \in \mathcal{T}, \forall \mathbf{x}' \in \mathbb{N}^{\xi}, \mathbf{x}' \preceq \mathbf{x} \implies \mathbf{x}' \in \mathcal{T} \tag{3.12}$$

then, the set of minimal *linear separators for* $\mathcal{T}$ *and* $\bar{\mathcal{T}}$ *will contain a separator with only* non-negative *coefficients.*

Proof: [2] Suppose that the minimum number of linear inequalities with free coefficients needed to separate $\mathcal{T}$ from $\bar{\mathcal{T}}$ is k, and such a minimal separator is represented by the set of hyperplanes $H = \{h_i \equiv (\mathbf{A}[i,.], \mathbf{b}[i]), i \in \{1, \ldots, k\}\}$. We need to show that there exists a linear separator with only non-negative coefficients that achieves the separation of these two sets, and the number of inequalities employed by this new separator is also k.

If H happens to satisfy the posed non-negativity requirement, then there is nothing left to be proved. Otherwise, consider an inequality $h_i = (\mathbf{A}[i,.], \mathbf{b}[i])$ employed in H that violates the posed non-negativity requirement. First, we notice that since (i) all inequalities employed by separator H are of the '$\leq$' type (according to Definition 3.1), and (ii) $\mathbf{0}$ satisfies these inequalities (by Eq. 3.12), the right-hand-side coefficient $\mathbf{b}[i]$ of the considered inequality $h_i = (\mathbf{A}[i,.], \mathbf{b}[i])$ must be non-negative, i.e.,

$$\mathbf{b}[i] \geq 0 \tag{3.13}$$

Second, from the definition of minimal linear separation (c.f. Def. 3.1), we can infer that

$$\forall\ \mathbf{x} \in \mathcal{T}:\ \mathbf{A}[i,.] \cdot \mathbf{x} \leq \mathbf{b}[i] \quad \wedge \quad \exists\ \mathbf{y} \in \bar{\mathcal{T}}:\ \mathbf{A}[i,.] \cdot \mathbf{y} > \mathbf{b}[i] \tag{3.14}$$

[2] A first proof of this result was presented in [62]. Here, we present a briefer proof that was provided by Prof. Craig Tovey.

The first component in the expression of Equation 3.14 is a direct application of the definition of linear separation, while the second component is due to the minimality of separation (if this component was not valid, h_i can be removed from H without any consequences for the correctness of the classification process, and therefore, separator H is not minimal). Let $\bar{\mathcal{T}}^{(i)} \equiv \{\mathbf{y} : \mathbf{y} \in \bar{\mathcal{T}} \ \wedge \ \mathbf{A}[i,.] \cdot \mathbf{y} > \mathbf{b}[i]\}$. We shall use Equation 3.12 to show the existence of another hyperplane $h'_i = (\mathbf{A'}[i,.], \mathbf{b'}[i])$ with only non-negative coefficients which separates $\mathcal{T}$ and $\bar{\mathcal{T}}^{(i)}$. If such a hyperplane exists, it can replace the original hyperplane $h_i = (\mathbf{A}[i,.], \mathbf{b}[i])$ in separator H while maintaining a successful separation of sets $\mathcal{T}$ and $\bar{\mathcal{T}}$. Furthermore, by invoking the same argument for every inequality of H that violates the non-negativity requirement, we can obtain a linear separator H' that (i) employs the same number of inequalities, k, and (ii) it has only non-negative coefficients for all the inequalities involved; hence, this is the sought separator.

Construct the hyperplane $h'_i = (\mathbf{A'}[i,.], \mathbf{b'}[i])$ as follows:

$$\mathbf{b'}[i] = \mathbf{b}[i] \ \wedge \ \mathbf{A'}[i,j] = \begin{cases} 0, & \text{if } \ \mathbf{A}[i,j] < 0 \\ \mathbf{A}[i,j], & \text{o.w.} \end{cases} \tag{3.15}$$

Let $\mathbf{x}^*$ be a vector in $\mathcal{T}$. Similar to Equation 3.15, construct the vector $\mathbf{x}_i^*$ as follows:

$$\mathbf{x}_i^*[j] = \begin{cases} 0, & \text{if } \ \mathbf{A}[i,j] < 0 \\ \mathbf{x}^*[j], & \text{o.w.} \end{cases} \tag{3.16}$$

Equation 3.12 implies that $\mathbf{x}_i^* \in \mathcal{T}$. Hence, Equations 3.14–3.16 imply that

$$\mathbf{A'}[i,.] \cdot \mathbf{x}^* = \mathbf{A}[i,.] \cdot \mathbf{x}_i^* \leq \mathbf{b}[i] = \mathbf{b'}[i] \ \wedge \ \forall \mathbf{y} \in \bar{\mathcal{T}}^{(i)}, \ \mathbf{A'}[i,.] \cdot \mathbf{y} \geq \mathbf{A}[i,.] \cdot \mathbf{y} > \mathbf{b}[i] \tag{3.17}$$

Since $\mathbf{x}^*$ is an arbitrary vector in $\mathcal{T}$, the sought result is established. □

The next proposition concretizes the previous results in the context of Garada RAS.

Proposition 3.4 *Given a Gadara RAS Φ_g, the set of minimal linear separators for the vectors sets corresponding to the subspaces S_{rs} and $S^b_{r\bar{s}}$ will always contain a separator $(\mathbf{A}, \mathbf{b})$ with non-negative coefficients $\mathbf{A}[i,j]$ and $\mathbf{b}[i]$.*

Proof: Lemma 2.1 and Corollary 3.1 show that S_{rs} and $S^b_{r\bar{s}}$ satisfy the conditions of Proposition 3.3. Therefore, applying Proposition 3.3 on the sets S_{rs} and $S^b_{r\bar{s}}$, the sought result is established. □

Converting the separation problem of $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ to an equivalent separation problem of reduced dimensionality As explained in Section 2.5, after thinning the sets of safe states and boundary unsafe states to their respective maximal and minimal elements, we have frequently encountered a situation where many components of the vectors included in the set $\hat{S}_{rs}$ are always greater than or equal to the corresponding components of the vectors included in the set $\hat{S}^b_{r\bar{s}}$. We shall see in the next proposition that dropping these dimensions does not compromise the feasibility or the optimality of the linear classification problem. In the statement and the proof of this proposition, we employ the relevant notation and terminology that were introduced in Section 2.5.

Proposition 3.5 *Given a Gadara RAS Φ_g, there exists a set of k hyperplanes, Q, that separates the projected sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$ in subspace V_P,* iff *there exists a set of k hyperplanes, H, that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ in the original space V.*

Proof: First we show that the existence of a linear separator Q with k inequalities for the projected sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$ in subspace V_P, implies the existence of a separator H with the same number of inequalities that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ in the original space V. Let $q_i = (\mathbf{A}^Q[i,.], \mathbf{b}^Q[i])$ be an arbitrary hyperplane of Q, and denote by $P(\hat{S}^b_{r\bar{s}})^{(i)}$ the set of points in $P(\hat{S}^b_{r\bar{s}})$ separated by q_i; i.e.,

$$\forall \mathbf{x} \in P(\hat{S}_{rs}) : \ \mathbf{A}^Q[i,.] \cdot \mathbf{x} \ \leq \ \mathbf{b}^Q[i] \quad \wedge \quad \forall \mathbf{y} \in P(\hat{S}^b_{r\bar{s}})^{(i)} : \ \mathbf{A}^Q[i,.] \cdot \mathbf{y} > \mathbf{b}^Q[i] \quad (3.18)$$

Also, let $(\hat{S}^b_{r\bar{s}})^{(i)} \subseteq \hat{S}^b_{r\bar{s}}$ be the set of states in $\hat{S}^b_{r\bar{s}}$ with their projection being in the set $P(\hat{S}^b_{r\bar{s}})^{(i)}$. To prove our case, it suffices to show that there exists a hyperplane h_i in the original space V that separates $(\hat{S}^b_{r\bar{s}})^{(i)}$ from $\hat{S}_{rs}$. This hyperplane $h_i = (\mathbf{A}^H[i,.], \mathbf{b}^H[i])$ can be constructed as follows:

$$\mathbf{b}^H[i] := \mathbf{b}^Q[i] \ \wedge \ \forall j \in L_0 : \ \mathbf{A}^H[i,j] := 0 \ \wedge \ \forall j \in L_P : \ \mathbf{A}^H[i,j] := \mathbf{A}^Q[i, \Pi(j)] \quad (3.19)$$

Indeed, we can see that $\forall \mathbf{x} \in \hat{S}_{rs}$,

$$\mathbf{A}^H[i,.] \cdot \mathbf{x} = \sum_{j \in L} \mathbf{A}^H[i,j] \cdot \mathbf{x}[j] = \sum_{j \in L_p} \mathbf{A}^H[i,j] \cdot \mathbf{x}[j] = \mathbf{A}^Q[i,.] \cdot \mathbf{x}_p \ \leq \ \mathbf{b}^Q[i] = \mathbf{b}^H[i] \quad (3.20)$$

where $\mathbf{x}_p$ is the image of $\mathbf{x}$ in subspace V_P. Similarly we have $\forall \mathbf{y} \in (\hat{S}^b_{r\bar{s}})^{(i)}$,

$$\mathbf{A}^H[i,.] \cdot \mathbf{y} = \sum_{j \in L} \mathbf{A}^H[i,j] \cdot \mathbf{y}[j] = \sum_{j \in L_p} \mathbf{A}^H[i,j] \cdot \mathbf{y}[j] = \mathbf{A}^Q[i,.] \cdot \mathbf{y}_p > \mathbf{b}^Q[i] = \mathbf{b}^H[i] \quad (3.21)$$

where $\mathbf{y}_p$ is the image of $\mathbf{y}$ in subspace V_P. Therefore h_i separates $(\hat{S}^b_{r\bar{s}})^{(i)}$ from $\hat{S}_{rs}$, and the forward part of Proposition 3.5 is proved.

Next, we show the validity of the reverse part of the proposition, i.e., that the existence of a linear separator H with k inequalities that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ in the original space V, implies the existence of a linear separator Q with the same number of inequalities that separates the projected sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$ in subspace V_P. To prove this result, we first notice that Proposition 3.4 implies that if there exists a linear separator H with k inequalities that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ in the original space V, then there also exists a linear separator H' with k inequalities and non-negative coefficients that separates the same two sets in V. So, in the following we shall focus on such a separator H'. The rest of the proof proceeds similarly to the proof provided for the forward part of the proposition, but it also relies on the fact that

$$\forall \mathbf{s} \in \hat{S}_{rs}, \ \forall \mathbf{u} \in \hat{S}^b_{r\bar{s}}, \ \forall j \in L_0 : \ \mathbf{s}[j] \geq \mathbf{u}[j] \quad (3.22)$$

Let $h_i = (\mathbf{A}^{H'}[i,.], \mathbf{b}^{H'}[i]) \in H'$, and denote by $(\hat{S}^b_{r\bar{s}})^{(i)}$ the set of points in $(\hat{S}^b_{r\bar{s}})$ separated by h_i. Then,

$$\forall \mathbf{x} \in \hat{S}_{rs} : \ \mathbf{A}^{H'}[i,.] \cdot \mathbf{x} \ \leq \ \mathbf{b}^{H'}[i] \ \ \wedge \ \ \forall \mathbf{y} \in (\hat{S}^b_{r\bar{s}})^{(i)} : \ \mathbf{A}^{H'}[i,.] \cdot \mathbf{y} > \mathbf{b}^{H'}[i] \tag{3.23}$$

Also, let $P(\hat{S}^b_{r\bar{s}})^{(i)} \subseteq P(\hat{S}^b_{r\bar{s}})$ be the projection of $(\hat{S}^b_{r\bar{s}})^{(i)}$ on subspace V_P. We need to show that there exists a hyperplane q_i in subspace V_P which separates $P(\hat{S}^b_{r\bar{s}})^{(i)}$ from $P(\hat{S}_{rs})$. This hyperplane $q_i = (\mathbf{A}^Q[i,.], \mathbf{b}^Q[i])$ can be constructed as follows:

$$\mathbf{b}^Q[i] := \mathbf{b}^{H'}[i] - min_{\mathbf{s} \in \hat{S}_{rs}} \{ \sum_{j \in L_0} \mathbf{A}^{H'}[i,j] \cdot \mathbf{s}[j] \} \ \ \wedge \ \ \forall j \in L_P : \ \mathbf{A}^Q[i, \Pi(j)] = \mathbf{A}^{H'}[i,j] \tag{3.24}$$

Notice that Equation 3.23 together with the non-negativity of the coefficients of the separator H' imply that $b^Q[i] \geq 0, \ \forall i$. For the separator of Equation 3.24, we can see that

$$\forall \mathbf{x}_p \in P(\hat{S}_{rs}) : \ \mathbf{A}^Q[i,.] \cdot \mathbf{x}_p = \sum_{j \in L_P} \mathbf{A}^{H'}[i,j] \cdot \mathbf{x}[j] \ \leq$$
$$\mathbf{b}^{H'}[i] - \sum_{j \in L_0} \mathbf{A}^{H'}[i,j] \cdot \mathbf{x}[j] \leq \ \mathbf{b}^{H'}[i] - min_{\mathbf{s} \in \hat{S}_{rs}} \{ \sum_{j \in L_0} \mathbf{A}^{H'}[i,j] \cdot \mathbf{s}[j] \} = \mathbf{b}^Q[i] \tag{3.25}$$

where $\mathbf{x} \in \hat{S}_{rs}$ is an element in the pre-image of $\mathbf{x}_p$ in the original space V. Similarly, we can see that

$$\forall \mathbf{y}_p \in P(\hat{S}^b_{r\bar{s}})^{(i)} : \ \mathbf{A}^Q[i,.] \cdot \mathbf{y}_p = \textstyle\sum_{j \in L_p} \mathbf{A}^{H'}[i,j] \cdot \mathbf{y}[j] >$$
$$\mathbf{b}^{H'}[i] - \textstyle\sum_{j \in L_0} \mathbf{A}^{H'}[i,j] \cdot \mathbf{y}[j] \geq \mathbf{b}^{H'}[i] - max_{\mathbf{u} \in \hat{S}^b_{r\bar{s}}} \{ \textstyle\sum_{j \in L_0} \mathbf{A}^{H'}[i,j] \cdot \mathbf{u}[j] \} \geq$$
$$\mathbf{b}^{H'}[i] - min_{\mathbf{s} \in \hat{S}_{rs}} \{ \textstyle\sum_{j \in L_0} \mathbf{A}^{H'}[i,j] \cdot \mathbf{s}[j] \} = \mathbf{b}^Q[i] \tag{3.26}$$

where $\mathbf{y} \in (\hat{S}^b_{r\bar{s}})^{(i)}$ is an element in the pre-image of $\mathbf{y}_p$ in the original space V, and the last inequality above holds true because of Equation 3.22 and the presumed non-negativity of the elements of $\mathbf{A}^{H'}$. Therefore, q_i separates $P(\hat{S}^b_{r\bar{s}})^{(i)}$ from $P(\hat{S}_{rs})$, and the proof is complete. □

The practical implication of Proposition 3.5 is that we can construct a minimal linear separator H for the sets $\hat{S}^b_{r\bar{s}}$ and $\hat{S}_{rs}$ by first developing a linear separator Q

for the projected sets $P(\hat{S}^b_{r\bar{s}})$ and $P(\hat{S}_{rs})$, and subsequently constructing H from Q through Equation 3.19, which is repeated here for emphasis and convenience:

$$\mathbf{b}^H[i] := \mathbf{b}^Q[i] \ \wedge$$
$$\forall j \in L_0 : \ \mathbf{A}^H[i,j] := 0 \ \ \wedge \ \ \forall j \in L_P : \ \mathbf{A}^H[i,j] := \mathbf{A}^Q[i, \Pi(j)] \tag{3.27}$$

Furthermore, the second part of the proof of Proposition 3.5 guarantees that we can have a *minimal* linear separator Q for the projected sets $P(\hat{S}^b_{r\bar{s}})$ from $P(\hat{S}_{rs})$ with *non-negative* coefficients.

Some further simplifications As explained in Section 2.5, the projection P might introduce some redundancy and dominance among the elements of $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$. Hence, these effects are identified, and subsequently, removed from each set, resulting in the generation of the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$. The fact that this additional thinning does not compromise the effectiveness of the obtained separator Q with respect to the separation of the sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$, can be argued on the basis of the non-negativity of the coefficients of the target separator Q; cf. Proposition 3.2.

3.4 Synthesizing the linear classifier Q through Mathematical Programming

The MIP formulation In this section we provide a Mixed Integer Programming (MIP) formulation [92] for the construction of the separator Q, that was specified in Section 3.3. We remind the reader that the primary inputs to this formulation are:

- the elements $\mathbf{x}_i$ of the projected safe state set $\widehat{P(\hat{S}_{rs})}$, and
- the elements $\mathbf{y}_i$ of the projected unsafe state set $\widehat{P(\hat{S}^b_{r\bar{s}})}$.

In the subsequent discussion, we shall set $m_s \equiv |\widehat{P(\hat{S}_{rs})}|$, $m_u \equiv |\widehat{P(\hat{S}^b_{r\bar{s}})}|$, and $n \equiv |L_P|$, i.e., n denotes the dimensionality of the subspace V_P supporting the vectors $\mathbf{x}_i$ and $\mathbf{y}_i$. Some additional inputs that parameterize this last stage of our design process, and provide additional controls to it, are as follows:

- A parameter w which provides an upper bound for the "size" of – i.e., the number of inequalities employed by – the sought separator. Such an upper bound is readily obtained as $w = m_u$ from Corollary 3.1 when combined with the results of Proposition 3.4 and Equation 3.27. A tighter value for w can be effectively computed through the heuristic discussed in the next section.

- A strictly positive parameter ϵ that controls the minimum distance of the points $\mathbf{y}_i$ from the separating hyperplanes, and should be set sufficiently close to zero. This parameter can be perceived as a "degree of separation" that is enforced between the two sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$. In order to guarantee that the employed value of ϵ is not unnecessarily large to the extent that it compromises the minimality of the derived solution, one should re-solve the proposed formulation for a sequence $\{\epsilon_i\}$ such that $\epsilon_i \to 0^+$, and consider the stability of the size of the obtained supervisors.

- A strictly positive parameter M that is useful for the modelling of the separation logic in the proposed MIP formulation, and must take sufficiently large values. This is the notorious "big-M" parameter that appears in many MIP formulations. A more detailed discussion on its role in the proposed formulation, as well as on its appropriate pricing, is provided in later parts of this section.

The variables employed by the proposed formulation are as follows:

- z_l, $l = 1, \ldots, w$, is a binary variable that is set to one if the l-th inequality is used for separation and to zero otherwise.

- $(\mathbf{A}[l,.], \mathbf{b}[l])$, $l = 1, \ldots, w$, are the coefficients to be employed by the l-th separating linear inequality.

- δ_{il}, $i = 1, \ldots, m_u$, $l = 1, \ldots, w$, is a binary variable that can be set to one only if the state $\mathbf{y}_i$ and the hyperplane $(\mathbf{A}[l,.], \mathbf{b}[l])$ violate the inequality $\mathbf{A}[l,.] \cdot \mathbf{y}_i \leq \mathbf{b}[l]$,

i.e., $\mathbf{y}_i$ is separated by $(\mathbf{A}[l,.], \mathbf{b}[l])$.

Finally, the formulation itself takes the following form:

$$\min \sum_{l=1}^{w} z_l \tag{3.28}$$

$$\forall i \in \{1,\ldots,m_s\},\ \forall l \in \{1,\ldots,w\} :\ \mathbf{A}[l,.] \cdot \mathbf{x}_i - \mathbf{b}[l]\ \leq\ 0 \tag{3.29}$$

$$\forall i \in \{1,\ldots,m_u\},\ \forall l \in \{1,\ldots,w\} :\ \mathbf{A}[l,.] \cdot \mathbf{y}_i - \mathbf{b}[l] + M \cdot (1 - \delta_{il}) \geq\ \epsilon \tag{3.30}$$

$$\forall i \in \{1,\ldots,m_u\} :\ \sum_{l=1}^{w} \delta_{il}\ \geq\ 1 \tag{3.31}$$

$$\forall l \in \{1,\ldots,w\},\ \forall j \in \{1,\ldots,n\} :\ 0\ \leq\ \mathbf{A}[l,j]\ \leq\ z_l \tag{3.32}$$

$$\forall l \in \{1,\ldots,w\}\ :\ 0\ \leq\ \mathbf{b}[l]\ \leq\ z_l \tag{3.33}$$

$$\forall i \in \{1,\ldots,m_u\},\ \forall l \in \{1,\ldots,w\} :\ \delta_{il} \in \{0,1\} \tag{3.34}$$

$$\forall l \in \{1,\ldots,w\} :\ z_l \in \{0,1\} \tag{3.35}$$

The validity of the above MIP formulation as a construction tool for the sought separator Q can be established as follows: First, the reader should notice that Equation 3.28 defines the objective of the formulation as the minimization of the number of hyperplanes that will be used for the pursued separation. Also, Equations 3.34 and 3.35 respectively enforce the binary nature of the variables δ_{il} and z_l. On the other hand, the constraints of Equations 3.32– 3.33 enforce (i) the non-negativity of

the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ in the returned solution, and also (ii) the requirement that the coefficients of the unused inequalities should be set to zero. An additional implication of the constraints expressed by the right inequalities in Equations 3.32–3.33, is the restriction of all the elements of the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ to values no greater than one. This effect does not compromise the generality of the obtained solution, since these elements can always be normalized to have values no greater than one. On the other hand, we shall see in the following discussion that the restriction of the elements of the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ in the interval $[0,1]$ enables the effective resolution of some other aspects of the formulation.

The separation logic is primarily expressed by the constraints of Equations 3.29–3.31. More specifically, the constraints of Equation 3.29 force every point corresponding to a safe vector $\mathbf{x}_i$, $i=1,\ldots,m_s$, to lie below each separating hyperplane. On the other hand, the constraints of Equations 3.30 and 3.31 require that every point corresponding to an unsafe vector $\mathbf{y}_i$, $i=1,\ldots,m_u$, lies above of at least one of the separating hyperplanes. To understand the detailed mechanism that enforces this requirement, first notice that for any vector $\mathbf{y}_i$ and inequality $(\mathbf{A}[l,.],\mathbf{b}[l])$ such that $\mathbf{A}[l,.]\cdot\mathbf{y}_i \geq \mathbf{b}[l]+\epsilon$, Equation 3.30 is satisfied irrespective of the value of the binary variable δ_{il} (provided that $M \geq 0$). In the opposite case, the corresponding variable δ_{il} must be set to zero, in order to attain the satisfaction of Equation 3.30 (provided that the non-negative parameter M is sufficiently large). But at the same time, Equation 3.31 requires that, for every point $\mathbf{y}_i$, at least one of the corresponding variables δ_{il}, $l=1,\ldots,w$, must be equal to one. Therefore, in any feasible solution of the proposed formulation, every point $\mathbf{y}_i$ must be above at least one of the hyperplanes $(\mathbf{A}[l,.],\mathbf{b}[l])$.

The above discussion also reveals the condition for the proper pricing of the parameter M:

$$M \;\geq\; \sup\{[\mathbf{A}[l,.]\cdot\mathbf{y}_i - \mathbf{b}[l] - \epsilon]^-\} \tag{3.36}$$

where $[a]^- \equiv |\min\{0, a\}|$, and the supremum is taken over all pairs of vectors $\mathbf{y}_i$ and inequalities ($\mathbf{A}[l,.]$, $\mathbf{b}[l]$) in any viable solution. An upper bound for the quantity on the right-hand-side of Equation 3.36 can be obtained by setting $\mathbf{A}[l,.] \cdot \mathbf{y}_i = 0$, and considering an upper bound for $\sup\{\mathbf{b}[l] + \epsilon\}$ which is bounded above by $1 + \epsilon$. Hence, M can be set equal to $1 + \epsilon$ during the solution of the considered formulation.

Finally, we would like to mention that we tried some symmetry breaking technique to simplify the solution of the formulation of Equations 3.28–3.35. More specifically, we imposed an order among the indicator variables $\{z_l\}$, i.e., $z_1 \geq \ldots \geq z_w$. However, this symmetry breaking did not improve the computational results provided in Section 3.6.

Example 3.1 The application of the formulation of Equations 3.28–3.35 to the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$ provided in Table 2.1, while setting $\epsilon = 0.01$, resulted in a linear classifier that is expressed by the following two inequalities:

$$\mathbf{s}[1] + \mathbf{s}[5] \leq 1.0$$

$$0.01 \cdot \mathbf{s}[1] + 0.99 \cdot \mathbf{s}[2] + \mathbf{s}[4] \leq 1.0$$

Indeed, the reader can verify that this system of inequalities is satisfied by every vector in $\widehat{P(\hat{S}_{rs})}$ and it is violated by every vector in $\widehat{P(\hat{S}^b_{r\bar{s}})}$. But then, the previous developments in this chapter imply that the one-step-lookahead policy defined by the above two inequalities is an effective implementation of the maximally permissive DAP for the RAS of Table 2.1. Furthermore, by its construction, this implementation is minimal, i.e., it uses the minimum possible number of linear inequalities that can represent effectively the maximally permissive DAP for the considered RAS.

Complexity considerations The MIP formulation of Equations 3.28–3.35 involves $(m_u + 1) \cdot w$ binary variables, $(|L_P| + 1) \cdot w$ real variables and $w \cdot (m_s + m_u + |L_P|) + m_u$ technological constraints.[3] Hence, the size of this formulation, in terms

[3]By technological constraints we mean all the formulation constraints except from those that

of the variables and constraints involved, is polynomially related to the size of the classified sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$ and the dimensionality of their supporting subspace V_P. In Section 3.6 we provide a set of computational results that demonstrate the tractability of the MIP formulation of Equations 3.28–3.35 for RAS instantiations from the Gadara RAS class with size comparable to those encountered in many practical applications. These results make us believe that the MIP formulation of Equations 3.28–3.35 will be an effective computational tool for the synthesis of minimal linear separators Q for a very broad range of the Gadara RAS instantiations to be encountered in "real-life" applications.

Yet, a general MIP formulation is always an expensive (non-polynomial complexity) proposition from a computational standpoint. Furthermore, m_s and m_u, i.e., the cardinalities of the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$, in general will grow super-polynomially with respect to the size of the underlying RAS Φ_g. Therefore, in the next section, we present a heuristic that can provide a trade-off between the structural minimality of the obtained separator, as measured by the number of the involved inequalities, and the computational effort for its development. We also show that the relative size of the separators returned by this heuristic to the size of any minimal separator is no higher than $\ln|\widehat{P(\hat{S}^b_{r\bar{s}})}|$. Therefore, the classifiers provided by this heuristic are still quite compact and computationally efficient.[4]

3.5 An efficient heuristic for the synthesis of the linear classifier Q

The main idea that underlies the heuristic proposed in this section is to construct the sought separator Q one hyperplane at a time; in particular, at each iteration we

impose the nonnegative and the binary nature of the various variables. These are the constraints that are explicitly considered in the computations performed by any solution algorithm for the MIP formulation.

[4]Actually, the heuristic of Section 3.5 can also assist the solution of the MIP formulation of Equations 3.28–3.35 itself, e.g., by providing feasible solutions to this formulation.

consider the set of points in $\widehat{P(\hat{S}^b_{r\bar{s}})}$ that have not been separated yet from the set $\widehat{P(\hat{S}_{rs})}$ by any of the constructed hyperplanes, and we try to identify a hyperplane that will separate the maximum possible number of these points from $\widehat{P(\hat{S}_{rs})}$. Corollary 3.1 of Section 3.2 together with Propositions 3.4 and 3.5 of Section 3.3 imply that in the case of the Gadara RAS, which is the focus of this chapter, this iterative procedure will terminate in a finite number of iterations.

Next, we present a MIP formulation that can support the computation of the hyperplane sought at each of the iterations described in the previous paragraph. The input data for this formulation are:

- the elements $\mathbf{x}_i$ of the projected safe state set $\widehat{P(\hat{S}_{rs})}$;
- the elements $\mathbf{y}_i$ of the projected unsafe state set $\widehat{P(\hat{S}^b_{r\bar{s}})}$ that remain unseparated in the current iteration;
- strictly positive parameters ϵ and M that play a role similar to that played by the corresponding parameters in the MIP formulation of Equations 3.28–3.35.

The MIP synthesizes a linear inequality with non-negative coefficient that separates the maximum number of unseparated unsafe states from the set of safe states. Also, similar to the MIP formulation of Equations 3.28–3.35, we let $m_s \equiv |\widehat{P(\hat{S}_{rs})}|$, $m_u \equiv |\{\mathbf{y}_i\}|$, and $n \equiv |L_P|$. The variables employed by this new formulation are as follows:

- $(\mathbf{a}, b)$ are the coefficients of the generated hyperplane.
- δ_i, $i = 1, \ldots, m_u$, is a binary variable that is set to one *iff* the state $\mathbf{y}_i$, and the generated hyperplane violate the inequality $\mathbf{a} \cdot \mathbf{y}_i \leq b$ (i.e., point $\mathbf{y}_i$ is separated by the generated hyperplane).

The formulation itself is expressed by the following equations:

$$\max \sum_{i=1}^{m_u} \delta_i \tag{3.37}$$

$$\forall i \in \{1, \ldots, m_s\} : \ \mathbf{a} \cdot \mathbf{x}_i - b \leq 0 \tag{3.38}$$

$$\forall i \in \{1, \ldots, m_u\} : \ \mathbf{a} \cdot \mathbf{y}_i - b + (1 - \delta_i) \cdot M \geq \epsilon \tag{3.39}$$

$$\forall j \in \{1, \ldots, n\} : \ 0 \leq \mathbf{a}[j] \leq 1 \tag{3.40}$$

$$0 \leq b \leq 1 \tag{3.41}$$

$$\forall i \in \{1, \ldots, m_u\} : \ \delta_i \in \{0, 1\} \tag{3.42}$$

Equation 3.37 expresses the objective of the considered formulation as the maximization of the number of the unsafe points that will be separated by the generated hyperplane. Equation 3.38 forces all the safe state vectors $\mathbf{x}_i$ to lie below the generated hyperplane. On the other hand, Equation 3.39, in collaboration with Equation 3.37, enforces the correct pricing of the indicator variables δ_i, and therefore, it ensures the correct evaluation of the objective function for any given pricing of the design variables $(\mathbf{a}, b)$. The detailed mechanics of this enforcement are as follows: For a given unsafe state $\mathbf{y}_i$, if $\mathbf{a} \cdot \mathbf{y}_i - b < \epsilon$, the corresponding variable δ_i is forced to zero. On the other hand, if $\mathbf{a} \cdot \mathbf{y}_i - b \geq \epsilon$, Equation 3.39 allows δ_i to take any of its two possible values, but the objective stated in Equation 3.37 will force δ_i to one, in any optimal solution. Equations 3.40– 3.41 constrain the generated hyperplanes to have non-negative coefficients, and they also enforce a normalization of these coefficients similar to the normalization performed by the formulation of Equations 3.28–3.35. Equation 3.42 enforces the binary nature of the variables δ_i. Finally, we notice that for reasons similar to those explained in the discussion of the formulation of Equations 3.28–3.35, the parameter M of Equation 3.39 can be safely set equal to $1 + \epsilon$

Algorithm 3.1 A heuristic algorithm for the iterative construction of a linear separator for the state sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$.

Input:(i) the set of safe states $\widehat{P(\hat{S}_{rs})}$; (ii) the set of unsafe states $\widehat{P(\hat{S}^b_{r\bar{s}})}$; (iii) the parameters ϵ and M;
Output:(i) a list LL containing the coefficients of the generated linear inequalities $(\mathbf{A}[l,.],\mathbf{b}[l])$; (ii) the number of the linear inequalities in LL, k'

1: $k' := 0$; {Let *UnseparatedUnsafeStates* be the set of all unsafe states $\mathbf{y}_i \in \widehat{P(\hat{S}^b_{r\bar{s}})}$ that are not separated by any of the already generated hyperplanes; i.e., $\mathbf{A}[l,.] \cdot \mathbf{y}_i - \mathbf{b}[l] < \epsilon$ for every hyperplane $(\mathbf{a},b)$ in list LL.}
2: $UnseparatedUnsafeStates \leftarrow \widehat{P(\hat{S}^b_{r\bar{s}})}$;
3: **while** *UnseparatedUnsafeStates*$\neq \emptyset$ **do**
4: $k' := k' + 1$;
5: Generate a hyperplane $(\mathbf{a},b)$ by solving the MIP formulation of Equations 3.37-3.42 with input $\widehat{P(\hat{S}_{rs})}$ and *UnseparatedUnsafeStates*
6: Add $(\mathbf{a},b)$ to LL
7: **for** each $\mathbf{y}_i \in$ *UnseparatedUnsafeStates* **do**
8: **if** $(\mathbf{a} \cdot \mathbf{y}_i - b \geq \epsilon)$ **then**
9: Remove $\mathbf{y}_i$ from *UnseparatedUnsafeStates*
10: **end if**
11: **end for**
12: **end while**
13: Return LL and k'

during the solution of the formulation.

The formulation of Equations 3.37–3.42 employs $(|L_P|+1)$ real and m_u binary variables in $(m_s + m_u + |L_P|)$ technological constraints; therefore, the computational effort required for its solution is expected to be much smaller than the corresponding effort required for the solution of the exact formulation of Equations 3.28-3.35. Our computational results reveal that the relevant gains are so substantial, that the heuristic proposed in this section will tend to run much faster than the algorithm that solves the exact formulation, in spite of the fact that the formulation of Equations 3.37–3.42 is solved repeatedly in the context of this heuristic.

The complete procedure that utilizes the formulation of Equations 3.37–3.42 for the iterative construction of a linear separator for the state sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$,

is depicted in Algorithm 3.1. The next proposition establishes an upper bound to the potential sub-optimality of this algorithm.[5]

Proposition 3.6 *The number of the separating hyperplanes obtained by Algorithm 3.1 is at most* $K \ln |\widehat{P(\hat{S}^b_{r\bar{s}})}|$*, where* K *is the minimum number of hyperplanes that achieves separation.*

Proof: In order to simplify the notation employed in this proof, we shall set $U \equiv \widehat{P(\hat{S}^b_{r\bar{s}})}$, i.e., the set of unsafe states fed to Algorithm 3.1, and we shall also set $m_u \equiv |\widehat{P(\hat{S}^b_{r\bar{s}})}|$. In addition, $W(h)$ will denote the elements of U that are separated from the set of safe states, $\widehat{P(\hat{S}_{rs})}$, by any given hyperplane h. We claim that for each $U' \subseteq U$, there exists a hyperplane h that separates at least $|U'|/K$ states of U'; i.e.,

$$\forall U' \subseteq U,\ \exists h:\ |W(h) \cap U'| \geq |U'|/K \tag{3.43}$$

To see the validity of Equation 3.43, just notice that if it was not true, we would need more than K hyperplanes to separate U', and the same fact would be true for the separation of the superset U. But this contradicts the definition of K as the minimum number of hyperplanes that achieves the separation of U.

Next consider the execution of Algorithm 3.1, and let $U_i \subseteq U$ denote the set of unsafe states still not separated after i steps of the algorithm. Also, let h_{i+1} be the hyperplane generated at step $(i+1)$. Hence,

$$W(h_{i+1}) \cap U_i = U_i \backslash U_{i+1} \tag{3.44}$$

Since the considered algorithm maximizes $|W(h_{i+1}) \cap U_i|$, we can infer from Equations 3.43 and 3.44 that

$$\begin{aligned} |W(h_{i+1}) \cap U_i| = |U_i| - |U_{i+1}| \geq |U_i|/K \implies \\ |U_{i+1}| \leq |U_i|(1 - 1/K) \end{aligned} \tag{3.45}$$

[5]The content of this result and the arguments employed in its proof are similar to the results developed for the set cover problem [87].

So, by induction on i, we have that

$$|U_i| \ \leq \ (1-1/K)^i m_u \tag{3.46}$$

Choosing $i \ \geq \ K\ln(m_u)$, we see that

$$|U_i| \ \leq \ (1-1/K)^{K\ln(m_u)} m_u \ < \ e^{-\ln(m_u)} m_u \ = \ 1 \tag{3.47}$$

Thus, all the unsafe states in U must have been separated from the safe states in $\widehat{P(\hat{S}_{rs})}$ after $K\ln(m_u)$ steps. $\Box$

Example 3.2 Closing this section, we notice that the application of the heuristic of Algorithm 3.1 to the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$ of Table 2.1, while setting $\epsilon = 0.01$, resulted in a linear classifier consisting of two inequalities, and therefore, of minimal size. For completeness, we mention that the obtained inequalities were as follows:

$$\mathbf{s}[1] + 0.01 \cdot \mathbf{s}[4] + 0.99 \cdot \mathbf{s}[5] \leq 1.0$$

$$\mathbf{s}[2] + \mathbf{s}[4] \leq 1.99$$

3.6 Computational results

In this section we report the results from a series of computational experiments, in which we applied the DAP design methodology described in the earlier parts of this chapter upon a number of randomly generated instantiations of Gadara RAS. We remind the reader that according to the class definition, all resources in a Gadara RAS possess unit capacity. On the other hand, each of the generated instances was further specified by:

- The number of resources in the system; the range of this parameter was between 1 and 14.
- The number of process types in the system; the range of this parameter was between 2 and 14. Furthermore, in the considered experiments all process types

were assumed to have a simple linear structure, with the corresponding graphs $\mathcal{G}_j$ being simple paths (i.e., paths without any loops) for all j.

- The number of transitions in each process, with each transition corresponding to a single resource acquisition or a single resource release. The range of this parameter was between 2 and 14, but additional logic was applied to ensure a meaningful resource allocation sequence; hence, the eventual number of transitions appearing in every generated process differed by the originally specified number.[6] In particular, upon its initiation, a process was allocated randomly one of the system resources. At every subsequent transition, the process was either releasing one of its allocated resources, or it was acquiring one of the remaining resources. The association of any given transition with a resource release or a resource acquisition was equiprobable, except for the case where the process found itself possessing no resources; in that case the next transition was an acquisition with probability one. Similarly, the selection of the resource to be released by the process during a release transition, or the resource to be added to its current acquisitions during an acquisition transition, was determined equiprobably among the corresponding resource sets. Once the pre-specified number of transitions was determined as described above, the necessary number of release transitions was appended so that the process eventually returned all the acquired resources. Furthermore, in order to remain consistent with the RAS structure of Definition 1.1, all process stages resulting from the above construction that might correspond to zero resource allocation were identified and pruned from the process-defining sequence.

The employed RAS generator was encoded in Java, and it was compiled and linked by Java 1.6.0. Each generated RAS instance, Φ_g, was subjected to the DAP design

[6]The particular RAS dynamics adopted in the presented experiments were chosen so that they imitate closely the lock allocation and deallocation in real computer programs.

process described in Figure 2.1. The construction of the linear classifier itself was performed according to, both, the exact and heuristic approaches described respectively in Sections 3.4 and 3.5. In the case of the exact approach, we imposed a hard limit of 30 minutes (or 1800 secs) for the solution of the MIP formulation of Equations 3.28–3.35. For instances that were not completely solved within this time budget, the solver terminated prematurely and reported the best feasible solution identified up to that point. All our computational experiments were performed on a 2.66 GHz quad-core Intel Xeon 5430 processor with 6 MB of cache memory and 32 GB RAM; however, each job was single-threaded. The algorithms involved in the preprocessing stages of the proposed methodology were encoded in C++, compiled and linked by the GNU g++ compiler under Unix, while the MIP formulations of Equations 3.28–3.35 and Equations 3.37–3.42, that are employed respectively by the exact and the heuristic approaches, were solved through ILOG CPLEX 11.1 with ILOG Concert technology using C++.

Table 3.1 reports a representative sample of the results that we obtained in our experiments.[7] The reported cases are ordered in decreasing magnitude of the corresponding $|S_{rs}|$, the number of reachable safe states (second column in the presented table). The first column in Table 3.1 reports the dimensionality of the original state space corresponding to each listed configuration. Columns 3 – 9 report the cardinalities of the state subsets extracted through the various processing stages depicted in Figure 2.2, and also, the dimensionality reduction that was obtained through the projection P, discussed in the earlier parts of this section. Columns 10 and 11, entitled by k_{exact} and k_{heur}, report the number of linear inequalities in the solutions returned by the exact and the heuristic approach, respectively. Furthermore, the qualification [O] and [F] of the values reported in Column 10 indicates whether the

[7]This sample has been selected from data resulting by the application of the proposed methodology on more than 500 instances of the Gadara RAS generated according to the logic described in the earlier parts of this section.

Table 3.1: A sample of our experimental results for the construction of linear classifiers

ξ	$\lvert S_{rs}\rvert$	$\lvert S_{r\bar{s}}\rvert$	$\lvert S^b_{r\bar{s}}\rvert$	$\lvert\hat{S}_{rs}\rvert$	$\lvert\hat{S}^b_{r\bar{s}}\rvert = \widehat{\lvert P(\hat{S}^b_{r\bar{s}})\rvert}$	$\lvert L_P\rvert$	$\lvert P(\hat{S}_{rs})\rvert$	$\widehat{\lvert P(\hat{S}_{rs})\rvert}$	k_{exact}	k_{heur}	t_{thinn}	t_{exact}	t_{heur}
48	8,696,502	71,677	71,677	162,052	5	7	35	9	1 [O]	1	80	0	1
65	5,699,463	268,807	267,853	389,332	43	23	3,613	315	4 [O]	4	57	8	0
99	5,696,776	1,165,958	1,021,301	1,544,165	155	35	3,286	533	8 [F]	8	100	1,800	259
59	5,501,728	1,321,928	1,137,856	152,570	21	21	1,340	245	4 [O]	5	70	8	1
65	4,432,641	283,561	278,490	660,951	28	21	3,068	367	4 [O]	4	42	8	1
48	3,994,272	348,576	348,576	105,606	12	10	44	22	2 [O]	2	39	0	0
89	3,718,540	706,177	622,035	938,461	122	31	2,672	418	6 [F]	7	56	1,800	218
71	3,144,658	249,690	246,301	754,307	75	22	1,366	330	7 [F]	7	30	1,800	2
59	3,102,964	56,752	56,752	201,944	38	20	2,386	235	4 [O]	4	25	12	0
78	2,841,494	834,672	750,951	386,960	40	28	1,802	175	4 [O]	4	43	11	0
105	2,521,030	556,743	518,684	929,498	168	38	2,633	643	7 [F]	8	42	1,800	187
53	2,501,508	501,060	433,632	54,496	18	17	307	71	3 [O]	3	27	0	0
62	1,953,671	110,937	103,311	109,964	57	17	272	81	2 [O]	2	18	2	0
65	1,906,704	152,387	150,349	423,799	50	21	1,139	266	6 [O]	7	17	1,095	1
99	1,696,349	382,291	352,622	587,314	152	36	2,295	553	6 [F]	8	25	1,800	203
53	1,567,434	17,579	17,579	84,109	29	17	1,194	163	3 [O]	3	11	1	0
89	1,240,726	188,189	181,689	413,175	113	33	2,005	467	6 [F]	7	13	1,800	5
55	1,197,240	121,442	97,434	30,481	13	15	86	30	2 [O]	2	11	0	1
41	963,900	9,618	9,618	29,354	5	7	35	9	1 [O]	1	6	0	0
62	911,283	209,248	199,507	98,772	13	18	380	67	2 [O]	2	8	3	0

obtained solution was optimal or just a feasible one (this would happen if the solution algorithm was terminated prematurely, upon the exhaustion of the 30 minute budget). Finally, Columns 12, 13 and 14, respectively entitled by t_{thin}, t_{exact} and t_{heur}, report the amount of computing time (in seconds) that was required to execute (i) the pre-processing steps indicated in Figure 2.2, (ii) the construction of the linear classifier obtained by the exact approach, and (iii) the construction of the linear classifier obtained by the heuristic approach. The perusal of the data provided in Table 3.1 reveals very clearly

- the efficacy and the significance of the various set-"thinning" steps performed by the process depicted in Figure 2.2,
- the solvability of the exact MIP formulation of Section 3.4 for very large configurations from the considered RAS class, as a result of this "thinning" process, and also,

- the capability of the heuristic algorithm of Section 3.5 to provide separators that are (i) very efficient compared to those returned by the exact approach, and (ii) obtained in computational times that are significantly shorter than the times necessary for the solution of the exact MIP formulation.

3.7 Concluding remarks

As explained in Chapter 2, the approach that is pursued in this thesis perceives the maximally permissive DAP as a classification problem. In this chapter, we have addressed the classifier design problem in the context of Gadara RAS, and we have shown that the sought classifiers can take the convenient form of a set of linear inequalities. Furthermore, extensive numerical experimentation in the context of the aforementioned RAS class has confirmed the tractability of the approach and its ability to provide compact implementations of the maximally permissive DAP for RAS with a very large size and very large state spaces.

The methodology for the synthesis of maximally permissive and parsimonious DAPs developed in this chapter, can also be applied to the broader RAS classes defined in Chapter 1, but in that case, there is no guarantee for its completeness. More specifically, the MIP formulation of Section 3.4 may fail to identify any feasible solutions, and the iterations performed by the heuristic of Section 3.5 may reach a point where none of the remaining unsafe states will be linearly separable from the safe ones. Hence, in the next chapters, we shall seek the extension and the detailed implementation of the basic approach presented in Chapter 2 in a way that guarantees its completeness for RAS classes beyond that of the Gadara RAS.

CHAPTER IV

NON-LINEAR CLASSIFIERS

4.1 Introduction

In chapter 3, the classifier design problem was addressed in the context of the Gadara RAS. In this chapter, we extend the classifier design problem to handle any instance from the broader RAS class that was defined in Chapter 1. Relaxing the restriction imposed by the Gadara RAS, i.e., having resource types of non-unit capacity, results in state spaces that are not necessarily binary, and hence, the convex hull of the safe states might contain unsafe states. This leads to the inability of the linear classifiers to guarantee the effective representation of the maximally permissive DAP.[1] The following example demonstrates this possibility.

Example 4.1 The RAS configuration defined in Table 4.1 has two resource types, R_1 and R_2, each with capacity $C(R_i) = 2$. It also has two process types, J_1 and J_2, where each process type consists of two processing stages with each stage engaging a single resource type at the amount indicated by the corresponding coefficient. The RAS state s is defined by the 4-tuple (x_1, x_2, x_3, x_4) corresponding to the stages Ξ_{11}, Ξ_{12}, Ξ_{21} and Ξ_{22} respectively. It can be easily seen that $S_{rs} \equiv \{[0,0,0,0]^T,\ [1,0,0,0]^T,\ [0,1,0,0]^T,\ [0,0,1,0]^T,\ [0,0,0,1]^T,\ [2,0,0,0]^T,\ [0,0,2,0]^T,$ $[1,1,0,0]^T,\ [0,0,1,1]^T\}$, and $S^b_{r\bar{s}} \equiv \{[1,0,1,0]^T,\ [2,0,1,0]^T,\ [1,0,2,0]^T,\ [2,0,2,0]^T\}$. Since the process instances executing the stages Ξ_{12} and Ξ_{22} can immediately exit the system upon their completion, we can study the problem by considering only the

[1] Technically, this inability is established by the infeasibility of the mathematical programming formulations of Equations 3.28–3.35 for the design of the relevant linear classifiers. We also point out that the absence of a linear classifier for the maximally permissive DAP implies the further inability to express this policy in the standard PN modeling framework; the reader is referred to [70] for further discussion on this issue.

Table 4.1: The RAS considered in Example 4.1

Resource Types	$\{R_1, R_2\}$
Resource Capacities	$C(R_1) = C(R_2) = 2$
Process Types	$\{J_1, J_2\}$
Process Routes	$\mathcal{G}_1 : \quad R_1 \rightarrow 2.R_2$
	$\mathcal{G}_2 : \quad R_2 \rightarrow 2.R_1$

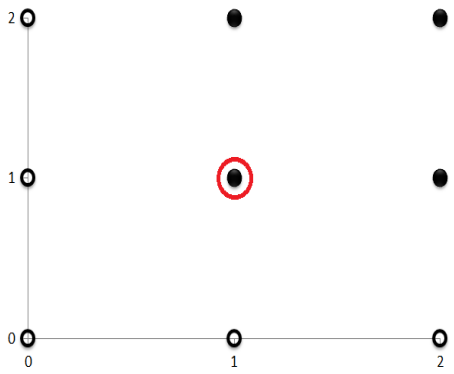


Figure 4.1: Characterization of the safe and unsafe reachable states for Example 4.1, in the projected sub-space defined by the state components x_1 and x_3; reachable safe states are depicted by white circles and reachable unsafe states by black ones.

projection of the reachable state space to the subspace defined by the components x_1 and x_3; Figure 4.1 provides this characterization. As it can be seen in the figure, the circled unsafe state lies in the convex hull of the set of safe states. Hence, the two sets can not be separated by the linear classifiers defined in Chapter 3.□

Inspired by the mathematical theory of artificial neural networks [63], in this chapter, we propose two classifier representations whose structures consist of two successive layers. The first classifier consists of two successive layers of linear inequalities, and hence, called *"Two-Layer Classifier"*. Figure 4.2 is a schematic diagram for the two-layer classifier. On the other hand, the second classifier consists of two different kinds of layers. More specifically, the first layer is a set of linear inequalities, whereas the second layer is a Boolean function that operates on the binary output of the first

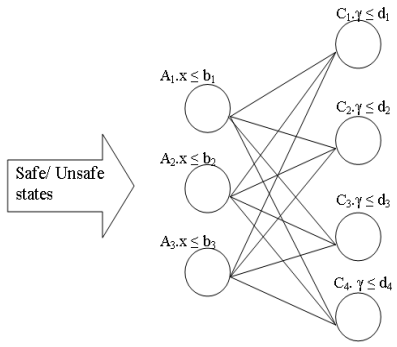


Figure 4.2: A schematic diagram of the two-layer classifier

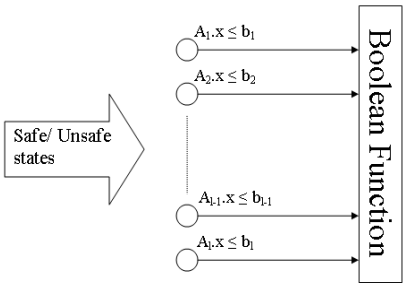


Figure 4.3: A schematic diagram of the Boolean classifier

layer. A classifier with such a structure is called "Boolean classifier". Figure 4.3 is a schematic diagram for the Boolean classifier.

In both classifiers, the role of the first layer of linear inequalities is the same. Basically a RAS state vector is evaluated against each linear inequality in the first layer, and the results are represented by a vector of binary indicators. Each indicator is associated with a linear inequality, and it indicates whether the inequality is satisfied or violated by the state vector. For the purposes of the proposed classification scheme, this vector of binary indicators can be perceived as the image of the original state vector under the transformation performed by the linear inequalities of the first layer. Applying the first-layer transformation, the problem is reproduced in the binary domain. In order to enable the separation between the safe and the unsafe states, the first-layer inequalities are constructed such that the image of the set of

safe states does not intersect with the image of the set of unsafe states.

In the light of the above ideas, the two-layer classifier is presented in Section 4.2, whereas the Boolean classifier is presented in Section 4.3. Finally, Section 4.4 concludes the chapter.

4.2 Two-layer classifiers

In this section, we show that for the considered RAS classes, the sought classifiers can take the form of two successive layers of linear inequalities, and we subsequently proceed to provide various methods for constructing the proposed classifiers. Hence, this section evolves as follows: Subsection 4.2.1 provides a formal definition of the two-layer classification problem, and subsequently, it proceeds to show the completeness of the adopted structure w.r.t. the addressed RAS classes. In Subsection 4.2.2, the two-layer classification problem is reduced to an equivalent problem with a much smaller input set in terms of the explicitly considered state vectors and their dimensionality. Subsection 4.2.3 addresses the synthesis of a two-layer classifier for the reduced classification problem, by formulating and solving this problem as a mixed integer program. On the other hand, Subsection 4.2.4 offers a set of heuristics for the synthesis of the sought classifier, that can be used in the case that the MIP formulation of Subsection 4.2.3 is deemed to be computationally too costly. Finally, Subsection 4.2.5 reports a series of computational experiments that demonstrates the proposed methodologies for the construction of two-layer classifiers, and shows the efficacy of these methodologies.

4.2.1 The two-layer classifier design problem

We shall proceed by first defining the constructs employed by the sought classifier, and subsequently, we shall formally introduce this classifier, and prove its capability of achieving the sought separation.

Definition 4.1 *Given a point* $\mathbf{x}$ *and a linear inequality* $\langle \mathbf{a}, b \rangle$*, we define:*

$$\gamma(\mathbf{x}, \mathbf{a}, b) = \begin{cases} 0, & \textit{if } \mathbf{a} \cdot \mathbf{x} \leq b \\ 1, & \textit{if } \mathbf{a} \cdot \mathbf{x} > b \end{cases}$$

We also define:

$$\gamma(\mathbf{x}, \begin{bmatrix} \mathbf{a_1}^T \\ \dots \\ \mathbf{a_k}^T \end{bmatrix}, \begin{bmatrix} b_1 \\ \dots \\ b_k \end{bmatrix}) = (\gamma(\mathbf{x}, \mathbf{a_1}, b_1), \dots, \gamma(\mathbf{x}, \mathbf{a_k}, b_k))^T$$

□

Hence, given a vector $\mathbf{x} \in \mathbb{N}^\xi$, a $k \times \xi$ real-valued matrix $\mathbf{A}$, and a vector $\mathbf{b} \in \Re^k$, $\gamma(\mathbf{x}, \mathbf{A}, \mathbf{b})$ can be seen as a transformation $\mathbb{N}^\xi \rightarrow \{0,1\}^k$. To simplify the notation, we shall refer to $\gamma(\mathbf{x}, \mathbf{A}, \mathbf{b})$ as $\gamma_{\mathbf{x}}$ when $\mathbf{A}$ and $\mathbf{b}$ are known. Next, we introduce the two-layer classifier.

Definition 4.2 *Consider two vector sets* G *and* H *from an* ξ*-dimensional vector space* V*. A two-layer classifier* TLC *is defined as a 4-tuple* $\langle \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d} \rangle$ *where* $\mathbf{A}$ *is a* $k_1 \times \xi$ *real-valued matrix,* $\mathbf{b}$ *is a vector in* $\Re^{k_1}$*,* $\mathbf{C}$ *is a* $k_2 \times k_1$ *real-valued matrix, and* $\mathbf{d}$ *is a vector in* $\Re^{k_2}$*. We say that* TLC *separates* G *and* H iff

$$\begin{aligned} &\forall \mathbf{g} \in G, \forall i \in \{1, \dots, k_2\} : \mathbf{C}[i, .] \cdot \gamma(\mathbf{g}, \mathbf{A}, \mathbf{b}) \leq \mathbf{d}[i] \ \wedge \\ &\forall \mathbf{h} \in H, \ \exists i^* \in \{1, \dots, k_2\} : \mathbf{C}[i^*, .] \cdot \gamma(\mathbf{h}, \mathbf{A}, \mathbf{b}) > \mathbf{d}[i^*] \end{aligned} \tag{4.1}$$

The size of the classifier, $|TLC|$*, is determined by the total number of operations required to classify a given vector. Thus,* $|TLC| = (2 \cdot \xi + 1) \cdot k_1 + (2 \cdot k_1 + 1) \cdot k_2$*.* □

The first layer of the classifier of Definition 4.2 implements the transformation $\gamma(., \mathbf{A}, \mathbf{b})$, whereas the second layer acts as a linear separator for the sets $I(G)$ and $I(H)$, where $I(G) \equiv \{\mathbf{z} \in \{0,1\}^k \mid \exists \mathbf{g} \in G \text{ s.t. } \mathbf{z} = \gamma(\mathbf{g}, \mathbf{A}, b)\}$ and a similar definition applies for $I(H)$.

To illustrate the size of the classifier, we notice that: (i) the first layer of the classifier $(\mathbf{A}, \mathbf{b})$ has k_1 linear inequalities, and (ii) to evaluate each of these linear inequalities, ξ multiplication operations, ξ addition operations and one comparison operation are required. Therefore, $(2 \cdot \xi + 1) \cdot k_1$ operations are required by the first layer to evaluate a given vector. Similar analysis can be applied to the second layer to see that $(2 \cdot k_1 + 1) \cdot k_2$ operations are required by the second layer to classify a given vector. Therefore, we can conclude that $(2 \cdot \xi + 1) \cdot k_1 + (2 \cdot k_1 + 1) \cdot k_2$ operations are required by a two-layer classifier to classify a given vector.

Finally, we notice that the linear classifier $\langle \mathbf{A}, \mathbf{b} \rangle$ is a special case of the two-layer classifier $\langle \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d} \rangle$ where the second layer $\langle \mathbf{C}, \mathbf{d} \rangle$ is given by the linear inequality $\sum_{i=1}^{k_1} \gamma_i \leq 0$.

Given the above definitions, the problem addressed in this section can be succinctly stated as follows:

Definition 4.3 – ***The two-layer classification problem:*** *Given a RAS* Φ*, construct a minimum-sized two-layer classifier for the vector sets corresponding to the subspaces* S_{rs} *and* $S_{r\bar{s}}$*, i.e., the reachable safe and the reachable unsafe states of the considered RAS* Φ*.* $\square$

Also, the following functions will be useful in the subsequent developments.

Definition 4.4 *Given the set of reachable states* S_r *of a RAS instance* Φ *with state dimensionality* ξ*, define*

- $CAP_i \equiv \begin{cases} 0, & i = 0 \\ \sup\{\mathbf{x} \in S_r : \mathbf{x}[i]\}, & i := 1, \ldots, \xi \end{cases}$
- $l(m) \equiv \sum_{j=0}^{m} CAP_j,\ m := 0, \ldots, \xi.$
- $v(i) \equiv m,\ i := 1, \ldots, \sum_{j=1}^{\xi} CAP_j \ \wedge \ \ l(m-1) < i \leq l(m).$ $\square$

In more natural terms, for every $i \in \{1,\ldots,\xi\}$, CAP_i represents the maximum number of job instances in processing stage i across all the reachable states $\mathbf{s} \in S_r$. CAP_0 is defined only for mathematical convenience. On the other hand, the functions $l(m),\ m \in \{0,\ldots,\xi\}$, and $v(i),\ i \in \{1,\ldots,\sum_{j=1}^{\xi} CAP_j\}$, essentially establish an indexing scheme that will be useful for the formal characterization of the sought classifier. More specifically, these two functions are employed in the statement and proof of the following lemma:

Lemma 4.1 *Consider a RAS instance* Φ *and the corresponding set of reachable states* S_r *whose vectors are of dimensionality* ξ*. Define the real matrix* $\mathbf{A}$ *and the vector* $\mathbf{b}$ *as follows:*

$$\forall i \in \{1,\ldots,\sum_{q=1}^{\xi} CAP_q\},\ \forall j \in \{1,\ldots,\xi\},\ \mathbf{A}[i,j] = \begin{cases} 1, & \textit{if}\ \ j = v(i) \\ 0, & o.w. \end{cases} \tag{4.2}$$

$$\forall i \in \{1,\ldots,\sum_{l=1}^{\xi} CAP_l\},\ \mathbf{b}[i] = i - l(v(i)-1) - 1 \tag{4.3}$$

Then,

$$\forall \mathbf{s}_1, \mathbf{s}_2 \in S_r,\quad \mathbf{s}_1 \neq \mathbf{s}_2 \Longrightarrow \gamma(\mathbf{s}_1, \mathbf{A}, \mathbf{b}) \neq \gamma(\mathbf{s}_2, \mathbf{A}, \mathbf{b}) \tag{4.4}$$

Proof: First we notice that Equations 4.2 and 4.3 are essentially a more compact and programmatic representation of the following structure:

$$\mathbf{A} = \begin{bmatrix} 10\ldots0 \\ 10\ldots0 \\ \ldots \\ 10\ldots0 \\ 01\ldots0 \\ \ldots \\ 01\ldots0 \\ \ldots \\ \ldots \\ 00\ldots1 \\ \ldots \\ 00\ldots1 \end{bmatrix} \quad \wedge \quad \mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ \ldots \\ CAP_1 - 1 \\ 0 \\ \ldots \\ CAP_2 - 1 \\ \ldots \\ \ldots \\ 0 \\ \ldots \\ CAP_\xi - 1 \end{bmatrix} \tag{4.5}$$

Next, consider two states $\mathbf{s}_1, \mathbf{s}_2 \in S_r$, such that $\mathbf{s}_1 \neq \mathbf{s}_2$. Therefore, there exists $i \in \{1, \ldots, \xi\}$ such that $\mathbf{s}_1[i] \neq \mathbf{s}_2[i]$. Without loss of generality, assume that $\mathbf{s}_1[i] > \mathbf{s}_2[i]$ (otherwise, simply reverse the roles of $\mathbf{s}_1$ and $\mathbf{s}_2$ in the subsequent argument). Then, consider the inequality defined by the k-th row of matrix $\mathbf{A}$ and the corresponding component of vector $\mathbf{b}$, where $k = \sum_{q=0}^{i-1} CAP_q + \mathbf{s}_2[i] + 1$. From the structure implied by Equation 4.5, it follows that $\forall \mathbf{s} \in S_r,\ \mathbf{A}[k,.] \cdot \mathbf{s} = \mathbf{s}[i]$ and $\mathbf{b}[k] = \mathbf{s}_2[i]$. Since it has been assumed that $\mathbf{s}_1[i] > \mathbf{s}_2[i]$, Definition 4.1 implies that $\gamma_{\mathbf{s}_1}[k] = 1$ and $\gamma_{\mathbf{s}_2}[k] = 0$, and our result has been established. □

The next example concretizes the constructs that were introduced in the previous lemma.

Example 4.2 Assume that for a given RAS Φ, $S_{rs} \equiv \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5\} \equiv \{[0,0]^T,\ [1,0]^T,\ [2,0]^T,\ [0,1]^T,\ [0,2]^T\}$ and $S_{r\bar{s}} \equiv \{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4\} \equiv \{[1,1]^T,\ [2,1]^T,\ [1,2]^T,\ [2,2]^T\}$. The corresponding set of reachable states S_r is plotted in Figure 4.4. We can see

that $CAP_1 = CAP_2 = 2$. Furthermore, the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ that are defined by Equations 4.2 and 4.3 (or, equivalently, by Equation 4.5) for this particular case, induce the following set of inequalities on the RAS state: $\{x_1 \leq 0,\ x_1 \leq 1,\ x_2 \leq 0,\ x_2 \leq 1\}$. These inequalities, when combined with the transformation introduced in Definition 4.1, result in the indicator vectors shown in Table 4.2. As established by Lemma 4.1, all the indicator vectors are different from each other. □

The next proposition plays a central role in this section.

Proposition 4.1 *Given a RAS instance* Φ *and the corresponding state sets* S_{rs} *and* $S_{r\bar{s}}$*, there exists a two-layer classifier that separates these two subspaces.*

Proof: Consider the pair $(\mathbf{A}, \mathbf{b})$ defined in Lemma 4.1 and the image sets $I(S_{rs})$, $I(S_{r\bar{s}})$ of the sets S_{rs} and $S_{r\bar{s}}$ that are defined by the pair $(\mathbf{A}, \mathbf{b})$ and the binary transformation introduced in Definition 4.1. Since both sets S_{rs} and $S_{r\bar{s}}$ are subsets of the reachable space S_r, Lemma 4.1 implies that $I(S_{rs}) \cap I(S_{r\bar{s}}) = \emptyset$. But then, Corollary 3.1 implies that there exists a set of linear inequalities $\langle \mathbf{C}, \mathbf{d} \rangle$ that linearly separates $I(S_{rs})$ and $I(S_{r\bar{s}})$, and the result is established. □

Example 4.1 (cont.) It can be easily checked in Table 4.2 that $I(S_{rs}) \equiv \{[0,0,0,0]^T,\ [1,0,0,0]^T,\ [1,1,0,0]^T,\ [0,0,1,0]^T,\ [0,0,1,1]^T\}$ and $I(S_{r\bar{s}}) \equiv \{[1,0,1,0]^T,\ [1,1,1,0]^T,\ [1,0,1,1]^T,\ [1,1,1,1]^T\}$, and, clearly, $I(S_{rs}) \cap I(S_{r\bar{s}}) = \emptyset$. A linear inequality that separates $I(S_{rs})$ and $I(S_{r\bar{s}})$ is: $2 \cdot \gamma_1 + \gamma_3 + \gamma_4 \leq 2$. □

4.2.2 Simplifying the two-layer classification problem

In order to cope with the huge cardinality of the state spaces involved in the considered RAS class, we utilize the thinning techniques introduced in Section 2.5. However, the employment of these techniques requires the restriction of the coefficients of the two-layer classifier to non-negative values. Next we show that the class of the two-layer classifiers that results from this restriction remains complete with respect to

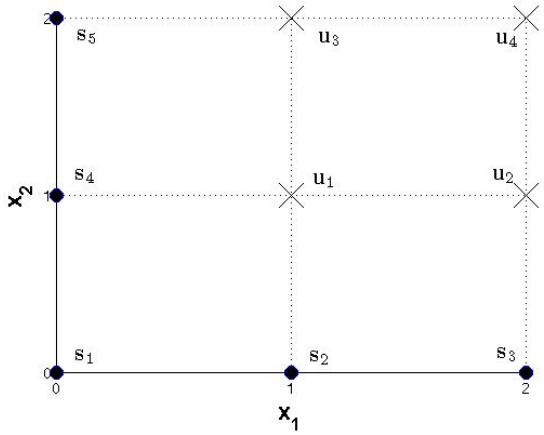


Figure 4.4: A plot of the sets S_{rs} and $S_{r\bar{s}}$ given in Example 4.2

Table 4.2: The indicators of the states of Example 4.2 w.r.t. the linear inequalities given at each column

	$x_1 \leq 0$	$x_1 \leq 1$	$x_2 \leq 0$	$x_2 \leq 1$
$\mathbf{s}_1$	0	0	0	0
$\mathbf{s}_2$	1	0	0	0
$\mathbf{s}_3$	1	1	0	0
$\mathbf{s}_4$	0	0	1	0
$\mathbf{s}_5$	0	0	1	1
$\mathbf{u}_1$	1	0	1	0
$\mathbf{u}_2$	1	1	1	0
$\mathbf{u}_3$	1	0	1	1
$\mathbf{u}_4$	1	1	1	1

the classification problem of Definition 4.3. The presented results are similar, in spirit, to their counterparts in Chapter 3, although the relevant proofs differ in their technicalities. [2]

We start with the following lemma that plays an important role in establishing the capability of the two-layer classifier with non-negative coefficients to achieve the sought separation. Furthermore, this lemma is crucial for the development of a set of

[2]On the other hand, contrary to the case considered in Chapter 3, we have not been able to provide a guarantee that the restriction imposed on the classifier coefficients will not affect the size of the minimal classifier.

heuristics that will be introduced in Subsection 4.2.4.

Lemma 4.2 *A two-layer classifier* $TLC = \langle \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d} \rangle$ *with* $\mathbf{C} \geq \mathbf{0}$ *and* $\mathbf{d} \geq \mathbf{0}$ *separates* S_{rs} *and* $S_{r\bar{s}}$ iff $\mathbf{A}$ *and* $\mathbf{b}$ *are constructed such that* $\nexists(\gamma_{\mathbf{s}} \in I(S_{rs}),\ \gamma_{\mathbf{u}} \in I(S_{r\bar{s}}))$ *with* $\gamma_{\mathbf{s}} \succeq \gamma_{\mathbf{u}}$.

Proof: ($\Longrightarrow$) Suppose that $\mathbf{A}$ and $\mathbf{b}$ are constructed such that $\nexists(\gamma_{\mathbf{s}} \in I(S_{rs}),\ \gamma_{\mathbf{u}} \in I(S_{r\bar{s}}))$ s.t. $\gamma_{\mathbf{s}} \succeq \gamma_{\mathbf{u}}$. Let $\tilde{I}(S_{rs}) \equiv \{\tilde{\gamma}_{\mathbf{s}} \in \{0,1\}^{k_1} \mid \exists \gamma_{\mathbf{s}'} \in I(S_{rs}) \text{ s.t. } \mathbf{0} \preceq \tilde{\gamma}_{\mathbf{s}} \preceq \gamma_{\mathbf{s}'}\}$. By assumption, $\tilde{I}(S_{rs}) \cap I(S_{r\bar{s}}) = \emptyset$. Also by construction, $\tilde{I}(S_{rs})$ contains all possible chains of integer vectors between the origin $\mathbf{0}$ and its maximal elements. Therefore, application of Corollary 3.1 and Proposition 3.3 on the sets $\tilde{I}(S_{rs})$ and $I(S_{r\bar{s}})$, implies the existence of a linear separator $\langle \mathbf{C}, \mathbf{d} \rangle$ with non-negative coefficients for the two sets. Since $I(S_{rs}) \subseteq \tilde{I}(S_{rs})$, we can see that $\langle \mathbf{C}, \mathbf{d} \rangle$ linearly separates $I(S_{rs})$ and $I(S_{r\bar{s}})$.

($\Longleftarrow$) Suppose that $\mathbf{A}$ and $\mathbf{b}$ are constructed such that $\exists(\gamma_{\mathbf{s}} \in I(S_{rs}),\ \gamma_{\mathbf{u}} \in I(S_{r\bar{s}}))$ s.t. $\gamma_{\mathbf{s}} \succeq \gamma_{\mathbf{u}}$. Since $\mathbf{C} \geq \mathbf{0}$, $\mathbf{C}\ \gamma_{\mathbf{u}} \leq \mathbf{C}\ \gamma_{\mathbf{s}} \leq \mathbf{d}$. Therefore, the separation can not be attained. □

Now we are ready to state and prove the main result of this section.

Proposition 4.2 *Given the two sets* S_{rs} *and* $S_{r\bar{s}}$ *for a RAS* Φ*, there exists a two-layer classifier with non-negative coefficients that separates the two subspaces.*

Proof: Construct the first layer $\langle \mathbf{A}, \mathbf{b} \rangle$ as in Equations 4.2, 4.3. It can be seen that

$$\forall \mathbf{s} \in S_r,\ \forall i \in \{1, \ldots \xi\},\ \mathbf{s}[i] = \sum_{j=l(v(i)-1)+1}^{l(v(i))} \gamma_{\mathbf{s}}[j] \tag{4.6}$$

For the sake of contradiction, assume that $\exists(\gamma_{\mathbf{s}} \in I(S_{rs}),\ \gamma_{\mathbf{u}} \in I(S_{r\bar{s}}))$ s.t. $\gamma_{\mathbf{s}} \succeq \gamma_{\mathbf{u}}$. Then, $\forall j_1, j_2 \in \{1, \ldots, k_1\}$ s. t. $j_1 \leq j_2$, $\sum_{j=j_1}^{j_2} \gamma_{\mathbf{s}}[j] \geq \sum_{j=j_1}^{j_2} \gamma_{\mathbf{u}}[j]$. Therefore, from Equation 4.6, $\forall i \in \{1, \ldots \xi\} : \mathbf{s}[i] \geq \mathbf{u}[i]$ where $\mathbf{s} \in S_{rs}$ and $\mathbf{u} \in S_{r\bar{s}}$; this result violates the monotonicity property of Proposition 2.1. Therefore, $\nexists(\gamma_{\mathbf{s}} \in I(S_{rs}),\ \gamma_{\mathbf{u}} \in$

$I(S_{r\bar{s}})$) s.t. $\gamma_{\mathbf{s}} \succeq \gamma_{\mathbf{u}}$. Applying Lemma 4.2 on $I(S_{rs})$ and $I(S_{r\bar{s}})$, we can see that there exists a linear separator $\langle \mathbf{C}, \mathbf{d} \rangle$ with non-negative coefficients for the two sets. Since, by their definition, $\mathbf{A} \geq \mathbf{0}$ and $\mathbf{b} \geq \mathbf{0}$, the sought result is established. □

From here on, we restrict our attention to two-layer classifiers with non-negative coefficients. Next, we establish the validity of the thinning operation introduced in Section 2.5 in the context of this classification problem. To this end, we introduce the following lemma:

Lemma 4.3 *Consider the vectors* $\mathbf{s}$, $\mathbf{s}'$, $\mathbf{u}$, *and* $\mathbf{u}'$, *and the two-layer classifiers* $TLC = \langle \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d} \rangle$ *and* $TLC' = \langle \mathbf{A}', \mathbf{b}', \mathbf{C}, \mathbf{d} \rangle$, *both with non-negative coefficients, and where each of the constructs* $\mathbf{A}$, $\mathbf{A}'$, $\mathbf{b}$, *and* $\mathbf{b}'$ *has* k_1 *rows. Also, assume that* $\mathbf{s}$ *and* $\mathbf{u}$ *are respectively classified by* TLC *as a safe and an unsafe state. Then:*

1. *If* $\forall i \in \{1, \ldots, k_1\}: \ \mathbf{A}'[i,.] \cdot \mathbf{s}' - \mathbf{b}'[i] \leq \mathbf{A}[i,.] \cdot \mathbf{s} - \mathbf{b}[i]$, *then* $\mathbf{s}'$ *is classified as a safe state by* TLC'.

2. *If* $\forall i \in \{1, \ldots, k_1\}: \ \mathbf{A}'[i,.] \cdot \mathbf{u}' - \mathbf{b}'[i] \geq \mathbf{A}[i,.] \cdot \mathbf{u} - \mathbf{b}[i]$, *then* $\mathbf{u}'$ *is classified as an unsafe state by* TLC'.

Proof: First, assume that $\forall i \in \{1, \ldots, k_1\}: \ \mathbf{A}'[i,.] \cdot \mathbf{s}' - \mathbf{b}'[i] \leq \mathbf{A}[i,.] \cdot \mathbf{s} - \mathbf{b}[i]$. Thus, $\gamma_{\mathbf{s}}[i] = 0 \Rightarrow \gamma'_{\mathbf{s}'}[i] = 0$. Therefore, $\gamma'_{\mathbf{s}'} \preceq \gamma_{\mathbf{s}}$. Since $\mathbf{s}$ is classified as a safe state by TLC, $\forall i \in \{1, \ldots, k_2\}: \ \mathbf{C}[i,.] \cdot \gamma_{\mathbf{s}} - \mathbf{d}[i] \leq 0$. Then the non-negativity of $\mathbf{C}$ implies that $\forall i \in \{1, \ldots, k_2\}: \ \mathbf{C}[i,.] \cdot \gamma'_{\mathbf{s}'} - \mathbf{d}[i] \leq \mathbf{C}[i,.] \cdot \gamma_{\mathbf{s}} - \mathbf{d}[i] \leq 0$. Therefore, $\mathbf{s}'$ is classified as a safe state by TLC'.

Next, assume that $\forall i \in \{1, \ldots, k_1\}: \ \mathbf{A}'[i,.] \cdot \mathbf{u}' - \mathbf{b}'[i] \geq \mathbf{A}[i,.] \cdot \mathbf{u} - \mathbf{b}[i]$. Thus, $\gamma_{\mathbf{u}}[i] = 1 \Rightarrow \gamma'_{\mathbf{u}'}[i] = 1$. Therefore, $\gamma'_{\mathbf{u}'} \succeq \gamma_{\mathbf{u}}$. Since $\mathbf{u}$ is classified as an unsafe state by TLC, $\exists i^* \in \{1, \ldots, k_2\}: \ \mathbf{C}[i^*,.] \cdot \gamma_{\mathbf{u}} - \mathbf{d}[i^*] > 0$. Then the non-negativity of $\mathbf{C}$ implies that $\mathbf{C}[i^*,.] \cdot \gamma'_{\mathbf{u}'} - \mathbf{d}[i^*] \geq \mathbf{C}[i^*,.] \cdot \gamma_{\mathbf{u}} - \mathbf{d}[i^*] > 0$. Therefore, $\mathbf{u}'$ is classified as an unsafe state by TLC'. □

Thinning the sets S_{rs} and $S^b_{r\bar{s}}$ by respectively focusing on their maximal and minimal elements Under the adopted non-negativity restriction for the parameters of the sought classifier, it is possible to obtain a two-layered classifier for the entire sets S_{rs} and $S^b_{r\bar{s}}$, by focusing the classifier design process only on the maximal elements of the first set, $\hat{S}_{rs}$, and the minimal elements of the second set, $\hat{S}^b_{r\bar{s}}$. This result is formally stated in the following proposition.

Proposition 4.3 *Any two-layer classifier $TLC = \langle \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d} \rangle$ with non-negative coefficients that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$, is also an effective separator for the entire sets S_{rs} and $S^b_{r\bar{s}}$.*

Proof: Let $\mathbf{s} \in S_{rs}$ be an arbitrary non-maximal reachable safe state vector, and $\mathbf{s}^* \in \hat{S}_{rs}$ be a maximal reachable safe state vector such that $\mathbf{s}^* \succ \mathbf{s}$.

Also, let $\mathbf{u} \in S^b_{r\bar{s}}$ be an arbitrary non-minimal boundary reachable unsafe state vector, and $\mathbf{u}^* \in \hat{S}^b_{r\bar{s}}$ be a minimal boundary reachable unsafe state vector such that $\mathbf{u}^* \prec \mathbf{u}$.

Assume that we have already identified a two-layer classifier $TLC = \langle \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d} \rangle$ s.t. $\mathbf{A} \geq \mathbf{0}$, $\mathbf{b} \geq \mathbf{0}$, $\mathbf{C} \geq \mathbf{0}$, and $\mathbf{d} \geq \mathbf{0}$, that separates $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$. Then, the non-negativity of all the coefficients of the linear inequalities, combined with the presumed relations of $\mathbf{s}$ to $\mathbf{s}^*$ and of $\mathbf{u}$ to $\mathbf{u}^*$, further imply that:

- $\forall i \in \{1, \ldots, k_1\}: \ \mathbf{A}[i,.] \cdot \mathbf{s} - \mathbf{b}[i] \leq \mathbf{A}[i,.] \cdot \mathbf{s}^* - \mathbf{b}[i]$. Hence, by Lemma 4.3, $\mathbf{s}$ is classified as a safe state.

- $\forall i \in \{1, \ldots, k_1\}: \ \mathbf{A}[i,.] \cdot \mathbf{u} - \mathbf{b}[i] \geq \mathbf{A}[i,.] \cdot \mathbf{u}^* - \mathbf{b}[i]$. Hence, by Lemma 4.3, $\mathbf{u}$ is classified as unsafe state.

Since states $\mathbf{s}$ and $\mathbf{u}$ were arbitrarily chosen, we can infer that the two-layer classifier $TLC = \langle \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d} \rangle$ is also an effective separator for the entire sets S_{rs} and $S^b_{r\bar{s}}$. $\square$

Converting the separation problem of $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ to an equivalent separation problem of reduced dimensionality As explained in Section 2.5, after thinning the sets of safe states and boundary unsafe states to their respective maximal and minimal elements, we have frequently encountered a situation where many components of the vectors included in the set $\hat{S}_{rs}$ are always greater than or equal to the corresponding components of the vectors included in the set $\hat{S}^b_{r\bar{s}}$. We shall see in this part of the section that dropping these dimensions does not compromise the feasibility or the optimality of the two-layer classification problem. In the statement and the proof of the next two propositions, we employ the relevant notation and terminology that were introduced in Section 2.5.

Proposition 4.4 *Consider a RAS instance Φ and its corresponding sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$. Furthermore, suppose that the two-layer classifier with non-negative coefficients $TLC^Q = \left\langle \mathbf{A}^Q, \mathbf{b}^Q, \mathbf{C}^Q, \mathbf{d}^Q \right\rangle$ separates the projected sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$ in subspace V_P. Then a two-layer classifier TLC^H for the original sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ is induced from TLC^Q according to the following equation:*

$$\begin{aligned} &\forall i,\ \forall j \in L_0 :\ \mathbf{A}^H[i,j] = 0 \ \wedge \ \ \forall i,\ \forall j \in L_P :\ \mathbf{A}^H[i,j] = \mathbf{A}^Q[i,\Pi(j)] \ \ \wedge \\ &\mathbf{b}^H = \mathbf{b}^Q \ \ \wedge \ \ \mathbf{C}^H = \mathbf{C}^Q \ \ \wedge \ \ \mathbf{d}^H = \mathbf{d}^Q \end{aligned} \tag{4.7}$$

Proof: To see the validity of Proposition 4.4, first notice that, under the stated assumptions,

$$\forall \mathbf{x} \in \hat{S}_{rs} \cup \hat{S}^b_{r\bar{s}},\ \forall i : \mathbf{A}^H[i,.] \cdot \mathbf{x} = \sum_{j \in L} \mathbf{A}^H[i,j] \cdot \mathbf{x}[j] = \sum_{j \in L_P} \mathbf{A}^H[i,j] \cdot \mathbf{x}[j] = \mathbf{A}^Q[i,.] \cdot \mathbf{x}^p$$

where $\mathbf{x}^p$ is the image of $\mathbf{x}$ in subspace V_P. Then, the result follows from the fact that $\mathbf{b}^H = \mathbf{b}^Q \ \ \wedge \ \ \mathbf{C}^H = \mathbf{C}^Q \ \ \wedge \ \ \mathbf{d}^H = \mathbf{d}^Q$ (c.f. Eq. 4.7). □

The practical implication of Proposition 4.4 is that we can construct a two-layer classifier TLC^H for the sets $\hat{S}^b_{r\bar{s}}$ and $\hat{S}_{rs}$ by first developing a two-layer classifier TLC^Q for the projected sets $P(\hat{S}^b_{r\bar{s}})$ and $P(\hat{S}_{rs})$, and subsequently constructing TLC^H from

TLC^Q through Equation 4.7. Furthermore, Equation 4.7 also implies that the construction of classifier TLC^H from classifier TLC^Q maintains the non-negativity of the coefficients, and therefore, it can be carried on the thinned sets $\hat{S}^b_{r\bar{s}}$ and $\hat{S}_{rs}$ and their projections, without compromising the validity of the derived classifiers for the broader sets of interest, $S^b_{r\bar{s}}$ and S_{rs}. It remains to establish that there will always exist a two-layer classifier with non-negative coefficients, TLC^Q, for the projected sets $P(\hat{S}^b_{r\bar{s}})$ and $P(\hat{S}_{rs})$; this is done by the following proposition.

Proposition 4.5 *Given a RAS Φ, there will always exist a two-layer classifier with non-negative coefficients, TLC^Q, that separates the projected sets $P(\hat{S}^b_{r\bar{s}})$ and $P(\hat{S}_{rs})$ in space V_P.*

Proof: First we notice that that the projected sets $P(\hat{S}^b_{r\bar{s}})$ and $P(\hat{S}_{rs})$ retain the monotonicity property of Proposition 2.1; in particular, there are no vectors $\mathbf{x}_1 \in P(\hat{S}_{rs})$ and $\mathbf{x}_2 \in P(\hat{S}^b_{r\bar{s}})$ such that $\mathbf{x}_2 \preceq \mathbf{x}_1$. Indeed, the existence of such a pair of vectors, $\mathbf{x}_1, \mathbf{x}_2$, when combined with the condition that defines projection P (c.f. Eq. 2.2), would further imply the existence of vectors $\mathbf{x'}_1 \in \hat{S}_{rs}$ and $\mathbf{x'}_2 \in \hat{S}^b_{r\bar{s}}$ such that $\mathbf{x'}_2 \preceq \mathbf{x'}_1$, and Proposition 2.1 would be violated.

Having established that the projected sets $P(\hat{S}^b_{r\bar{s}})$ and $P(\hat{S}_{rs})$ retain the monotonicity property of Proposition 2.1, the existence of a two-layer classifier with non-negative coefficients that separates these two sets can be established with an argument similar to that of Lemma 4.2 and Proposition 4.2. □

Some further simplifications As explained in Section 2.5, the projection P might introduce some redundancy and dominance among the elements of $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$. Hence, these effects are identified, and subsequently, removed from each set, resulting in the generation of the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$. The fact that this additional thinning does not compromise the effectiveness of the obtained separator TLC^Q with respect to the separation of the sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$, can be argued

using the same logic of Proposition 4.3. Finally, the image of these sets under the transformation of the first-layer linear inequalities will be denoted by $I(\widehat{P(\hat{S}_{rs})})$ and $I(\widehat{P(\hat{S}^b_{r\bar{s}})})$.

4.2.3 Synthesizing the two-layer classifier TLC^Q through Mathematical Programming

In this subsection, we provide a MIP formulation for the construction of the separator TLC^Q. In the subsequent discussion, let $m_s \equiv |\widehat{P(\hat{S}_{rs})}|$, $m_u \equiv |\widehat{P(\hat{S}^b_{r\bar{s}})}|$, $\Upsilon \equiv \widehat{P(\hat{S}_{rs})} \cup \widehat{P(\hat{S}^b_{r\bar{s}})}$, and $n \equiv |L_P|$. In addition to $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$, the following parameters are used to provide additional control to the MIP formulation:

- The parameter k_1 provides an upper bound for the number of inequalities employed by the first layer. Such an upper bound is readily obtained as $k_1 = \sum_{i=1}^{n} CAP_i$ from Proposition 4.2 where CAP_i is computed in the reduced subspace V_P. We shall refer to the term $\sum_{i=1}^{n} CAP_i$ as ν.

- The parameter k_2 provides an upper bound for the number of inequalities employed by the second layer. Such an upper bound is readily obtained as $k_2 = m_u$ in Section 3.4.

- Strictly positive parameters ϵ and $\{M_1, \ldots, M_6\}$ that play a role similar to that played by the corresponding parameters in the MIP formulation of Equations 3.28–3.35.

The variables employed by the proposed formulation are as follows:

- α_i, $i := 1, \ldots, k_1$, is a binary variable set to one *iff* the i-th linear inequality of the first layer is used for separation.

- β_i, $i := 1, \ldots, k_2$, is a binary variable set to one *iff* the i-th linear inequality of the second layer is used for separation.

- $(\mathbf{A}[i,.],\mathbf{b}[i])$, $i := 1,\ldots,k_1$, are the coefficients to be employed by the i-th linear inequality of the first layer.

- $(\mathbf{C}[i,.],\mathbf{d}[i])$, $i := 1,\ldots,k_2$, are the coefficients to be employed by the i-th linear inequality of the second layer.

- $\gamma_{\mathbf{x}}[i]$, $\mathbf{x} \in \Upsilon$, $i := 1,\ldots,k_1$, is a binary variable set to one *iff* $\mathbf{x}$ violates the linear inequality $\mathbf{A}[i,.] \cdot \mathbf{x} \leq \mathbf{b}[i]$.

- $\delta_{\mathbf{x}}[i]$, $\mathbf{x} \in \Upsilon$, $i := 1,\ldots,k_2$, is a binary variable set to one *iff* γ_x violates the linear inequality $\mathbf{C}[i,.] \cdot \gamma_x \leq \mathbf{d}[i]$.

- $\mathbf{r}_{\mathbf{x}}[i,j]$ is a real variable that is equal to $C[i,j] \cdot \gamma_{\mathbf{x}}[j]$.

- k_1^* is an integer variable that represents the total number of linear inequalities employed by the first layer.

- k_2^* is an integer variable that represents the total number of linear inequalities employed by the second layer.

- $k_{1\cdot 2}^*$ is an integer variable that is equal to $k_1^* \cdot k_2^*$.

- z_i, $i := 0,\ldots,k_1$, is an integer variable used to resolve the non-linearity of the term $k_1^* \cdot k_2^*$ as it will be revealed later.

Finally, the formulation itself takes the following form:

$$\text{Min } (2 \cdot n + 1) \cdot k_1^* + k_2^* + 2 \cdot k_{1\cdot 2}^* \tag{4.8}$$

$$\forall \mathbf{x} \in \{\Upsilon\},\ \forall i \in \{1,\ldots,k_1\} :\ \epsilon + M_1 \cdot (\gamma_{\mathbf{x}}[i] - 1) \leq \mathbf{A}[i,.] \cdot \mathbf{x} - \mathbf{b}[i] \leq M_2 \cdot \gamma_{\mathbf{x}}[i] \tag{4.9}$$

$$\forall \mathbf{x} \in \{\Upsilon\},\ \forall i \in \{1,\ldots,k_2\},\ \forall j \in \{1,\ldots,k_1\}:\ C[i,j]+\gamma_{\mathbf{x}}[j]-1 \leq \mathbf{r}_{\mathbf{x}}[i,j] \leq C[i,j] \tag{4.10}$$

$$\forall \mathbf{x} \in \{\Upsilon\},\ \forall i \in \{1,\ldots,k_2\},\ \forall j \in \{1,\ldots,k_1\}:\ 0 \leq \mathbf{r}_{\mathbf{x}}[i,j] \leq \gamma_{\mathbf{x}}[j] \tag{4.11}$$

$$\forall \mathbf{x} \in \{\Upsilon\},\ \forall i \in \{1,\ldots,k_2\}:\ \epsilon + M_3 \cdot (\delta_{\mathbf{x}}[i]-1) \leq \sum_{j=1}^{k_1} \mathbf{r}_{\mathbf{x}}[i,j] - \mathbf{d}[i] \leq M_4 \cdot \delta_{\mathbf{x}}[i] \tag{4.12}$$

$$\forall \mathbf{s} \in \widehat{P(\hat{S}_{rs})}:\ \sum_{i=1}^{k_2} \delta_{\mathbf{s}}[i] = 0 \tag{4.13}$$

$$\forall \mathbf{u} \in \widehat{P(\hat{S}^{b}_{r\bar{s}})}:\ \sum_{i=1}^{k_2} \delta_{\mathbf{u}}[i] \geq 1 \tag{4.14}$$

$$\forall i \in \{1,\ldots,k_1\},\ \forall j \in \{1,\ldots,n\}:\ 0 \leq \mathbf{A}[i,j] \leq \alpha_i \tag{4.15}$$

$$\forall i \in \{1,\ldots,k_1\}:\ 0 \leq \mathbf{b}[i] \leq \alpha_i \tag{4.16}$$

$$\forall i \in \{1,\ldots,k_2\},\ \forall j \in \{1,\ldots,k_1\}:\ 0 \leq \mathbf{C}[i,j] \leq \beta_i \tag{4.17}$$

$$\forall i \in \{1,\ldots,k_2\}:\ 0 \leq \mathbf{d}[i] \leq \beta_i \tag{4.18}$$

$$k_1^* = \sum_{i=1}^{k_1} \alpha_i \quad \wedge \quad k_2^* = \sum_{i=1}^{k_2} \beta_i \tag{4.19}$$

$$z_0 = 0 \quad \wedge \quad \forall i \in \{1,\ldots,k_1\}:\ 0 \leq\ z_i - z_{i-1}\ \leq k_2^* \tag{4.20}$$

$$\forall i \in \{1,\ldots,k_1\}: \quad k_2^* + M_5 \cdot (\alpha_i - 1) \leq \ z_i - z_{i-1} \ \leq M_6 \cdot \alpha_i \ \wedge \ k_{1\cdot 2}^* = z_{k_1} \qquad (4.21)$$

$$\forall i: \ \alpha_i, \ \beta_i, \ \gamma_{\mathbf{x}}[i], \ \delta_{\mathbf{x}}[i] \in \{0,1\}, \quad z_i \in {\mathbb{Z}_0}^+ \qquad (4.22)$$

The validity of the above MIP formulation as a construction tool for the sought classifier TLC^Q can be established as follows: First, the reader should notice that Equation 4.8 defines the objective of the formulation as the minimization of the size of the classifier. Also, Equation 4.22 enforces the binary nature of the variables α_i, β_i, $\gamma_{\mathbf{x}}[i]$, and $\delta_{\mathbf{x}}[i]$ and the discrete nature of z_i. On the other hand, the constraints of Equations 4.15 through 4.18 enforce (i) the non-negativity of the matrices $\mathbf{A}$ and $\mathbf{C}$ and the vectors $\mathbf{b}$ and $\mathbf{d}$ in the returned solution, and also (ii) the requirement that the coefficients of the unused inequalities should be set to zero. An additional implication of the constraints enforced through Equations 4.15– 4.18 is the restriction of all the elements of the matrices $\mathbf{A}$ and $\mathbf{C}$ and the elements of the vectors $\mathbf{b}$ and $\mathbf{d}$ to be no greater than one. This effect does not compromise the generality of the obtained solution(s) since these elements can always be normalized to have values no greater than one. The first-layer transformation is performed by Equation 4.9. In particular, we can see that by setting $\gamma_{\mathbf{x}}[i] = 0$, Equation 4.9 can be rewritten as $\epsilon - M_1 \leq \mathbf{A}[i,.] \cdot \mathbf{x} - \mathbf{b}[i] \leq 0$, whereas by setting $\gamma_{\mathbf{x}}[i] = 1$, it can be rewritten as $\epsilon \leq \mathbf{A}[i,.] \cdot \mathbf{x} - \mathbf{b}[i] \leq M_2$. Therefore, provided that M_1 and M_2 are large enough, and ϵ is small enough, if $\mathbf{A}[i,.] \cdot \mathbf{x} \leq \mathbf{b}[i]$, then $\gamma_{\mathbf{x}}[i]$ is forced to zero, whereas if $\mathbf{A}[i,.] \cdot \mathbf{x} > \mathbf{b}[i]$, then $\gamma_{\mathbf{x}}[i]$ is forced to one. In a similar way, Equation 4.12 sets the second layer indicators by forcing $\delta_{\mathbf{x}}[i]$ to zero if $\sum_{j=1}^{k_1} \mathbf{r}_{\mathbf{x}}[i,j] \leq \mathbf{d}[i]$, and to one if $\sum_{j=1}^{k_1} \mathbf{r}_{\mathbf{x}}[i,j] > \mathbf{d}[i]$.

The constraints of Equations 4.10 and 4.11 resolve the non-linearity of the term $C[i,j] \cdot \gamma_{\mathbf{x}}[j]$ by introducing the variable $\mathbf{r}_{\mathbf{x}}[i,j]$ to represent this non-linear term in

the following way: Equation 4.11 forces $\mathbf{r_x}[i,j]$ to zero if $\gamma_{\mathbf{x}}[j]$ equals zero. On the other hand, Equation 4.10 sets $\mathbf{r_x}[i,j] = C[i,j]$ if $\gamma_{\mathbf{x}}[j]$ equals one.

The constraints of Equation 4.13 enforce the requirement that the image of each safe state under the first-layer transformation should satisfy all the linear inequalities of the second layer, whereas the constraints of Equation 4.14 enforce the requirement that the image of each unsafe state under the first-layer transformation should violate at least one linear inequality of the second layer.

Equation 4.19 introduces the variables k_1^* and k_2^* as the number of linear inequalities employed by the first layer and the second layer, respectively.

Equations 4.20–4.21 are used to resolve the non-linearity of the term $k_1^* \cdot k_2^*$ by introducing the variable $k_{1.2}^*$ to represent this non-linear term. More specifically, $z_i - z_{i-1}$ equals zero if $\alpha_i = 0$, and equals k_2^* if $\alpha_i = 1$. Therefore, by setting $z_0 = 0$, we can see that $z_{k_1} = k_1^* \cdot k_2^*$. In particular, if $\alpha_i = 0$, Equation 4.21 can be rewritten as $k_2^* - M_5 \leq \ z_i - z_{i-1} \leq 0$; taking Equation 4.20 into account, we can see that $z_i - z_{i-1} = 0$ in this case. On the other hand, if $\alpha_i = 1$, Equation 4.21 can be rewritten as $k_2^* \leq \ z_i - z_{i-1} \ \leq M_6$; again taking Equation 4.20 into account, we can see that $z_i - z_{i-1} = k_2^*$ in this case.

The above discussion also reveals the condition for the proper pricing of the parameters $M_1, \ldots, M_6$. More specifically, the pricing of $\gamma_{\mathbf{x}}[j]$ implies that

$$M_1 \ \geq \ \sup\{[\mathbf{A}[i,.] \cdot \mathbf{x} - \mathbf{b}[i] - \epsilon]^-\} \tag{4.23}$$

where $[a]^- \equiv |\min\{0, a\}|$, and the supremum is taken over all pairs of vectors $\mathbf{x}$ and inequalities ($\mathbf{A}[i,.]$, $\mathbf{b}[i]$) in any viable solution. An upper bound for the quantity on the right-hand-side of Equation 4.23 can be obtained by setting $\mathbf{A}[i,.] \cdot \mathbf{x} = 0$, and considering an upper bound for $\sup\{b[i] + \epsilon\}$ which is bounded above by $1 + \epsilon$. Hence, M_1 can be set equal to $1 + \epsilon$.

Moreover, the pricing of $\gamma_{\mathbf{x}}[j]$ implies also that

$$M_2 \;\geq\; \sup\{\mathbf{A}[i,.]\cdot\mathbf{x}-\mathbf{b}[i]\} \tag{4.24}$$

where the supremum is taken over all pairs of vectors $\mathbf{x}$ and inequalities $(\mathbf{A}[i,.],\, \mathbf{b}[i])$ in any viable solution. An upper bound for the quantity on the right-hand-side of Equation 4.24 can be obtained by setting $\mathbf{b}[i]=0$, and considering an upper bound for $\sup\{\mathbf{A}[i,.]\cdot\mathbf{x}\}$. The boundedness of the vectors $\mathbf{x}$ and the matrix $\mathbf{A}$ implies that $\sup\{\mathbf{A}[i,.]\cdot\mathbf{x}\}\leq\nu$. Hence, M_2 can be set equal to ν.

Similar analysis can be applied to the pricing of $\delta_{\mathbf{x}}[i]$ to get that M_3 can be set equal to $1+\epsilon$ and that M_4 can be set equal to k_1. Equation 4.21 implies that M_5 and M_6 should be greater than or equal to k_2^*. But since k_2^* is not known a priori, and $k_2\geq k_2^*$, M_5 and M_6 can be set equal to k_2.

The next theorem provides a formal statement of the validity of the MIP formulation of Equations 4.8–4.22 as a classifier design tool for the classification problem considered in this section.

Theorem 4.1 *The application of the formulation of Equations 4.8–4.22 to the sets* $\widehat{P(\hat{S}_{rs})}$ *and* $\widehat{P(\hat{S}^b_{r\bar{s}})}$ *corresponding to a given RAS* Φ*, returns a minimum-sized two-layer classifier for these two sets, and through Equation 4.7, a minimum-sized two-layer classifier for the original sets* S_{rs} *and* $S^b_{r\bar{s}}$*.*

Complexity considerations The MIP formulation of Equations 4.8–4.22 involves $\mathcal{O}((m_u+m_s)\cdot(k_1+k_2)+n\cdot k_1+k_1\cdot k_2)$ binary variables, $\mathcal{O}(n\cdot k_1+(m_u+m_s)\cdot k_1\cdot k_2)$ real variables, $\mathcal{O}(k_1)$ integer variables, and $\mathcal{O}(n\cdot k_1+(m_u+m_s)\cdot k_1\cdot k_2)$ technological constraints. Hence, the size of this formulation, in terms of the variables and constraints involved, is polynomially related to the size of the classified sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$ and the dimensionality of their supporting sub-space V_P.

4.2.4 Two-stage synthesis of the two-layer classifier TLC^Q

In this subsection, we introduce an algorithm that constructs the sought classifier TLC^Q in two stages. In particular, the first stage involves the generation of the coefficients of the inequalities of the first layer of the classifier, whereas the second stage involves the generation of the coefficients of the inequalities of the second layer of the classifier. This approach reduces the complexity of the classifier synthesis and circumvents the complications resulting from the non-linearity incurred by the classifier construction through the MIP formulation of Subsection 4.2.3. On the other hand, the size of the obtained classifier might be larger. Lemma 4.2 plays a central role in this approach. According to Lemma 4.2, the separation is attained by TLC^Q *iff* $\mathbf{A}$ and $\mathbf{b}$ are constructed such that $\nexists(\gamma_{\mathbf{s}} \in I(\widehat{P(\hat{S}_{rs})}),\ \gamma_{\mathbf{u}} \in I(\widehat{P(\hat{S}^b_{r\bar{s}})})) :\ \gamma_{\mathbf{s}} \succeq \gamma_{\mathbf{u}}$. In other words, the sought separation is possible *iff*

$$\forall \mathbf{s} \in \widehat{P(\hat{S}_{rs})},\ \forall \mathbf{u} \in \widehat{P(\hat{S}^b_{r\bar{s}})},\ \exists i^* \in \{1, \ldots, k_1\} :$$
$$\gamma_{\mathbf{u}}[i^*] = 1\ \wedge\ \gamma_{\mathbf{s}}[i^*] = 0 \tag{4.25}$$

Algorithm 4.1 provides the outline of the two-stage approach. The goal of the first stage is to synthesize a minimal-cardinality set of linear inequalities such that Equation 4.25 is satisfied for each pair of safe and unsafe states. The next step is to obtain the image of the safe and the unsafe states under the first-layer transformation. By construction, these image vectors are binary vectors that possess the following monotonicity property between safe and unsafe states: $\nexists(\gamma_{\mathbf{s}} \in I(\widehat{P(\hat{S}_{rs})}),\ \gamma_{\mathbf{u}} \in I(\widehat{P(\hat{S}^b_{r\bar{s}})})) :$ $\gamma_{\mathbf{s}} \succeq \gamma_{\mathbf{u}}$. Therefore, we can utilize the MIP formulation given by Equations 3.28–3.35 in Section 3.4 to separate the image vectors. The following theorem formalizes these remarks.

Theorem 4.2 *The application of Algorithm 4.1 to the sets* $\widehat{P(\hat{S}_{rs})}$ *and* $\widehat{P(\hat{S}^b_{r\bar{s}})}$ *corresponding to a given RAS* Φ*, returns a two-layer classifier for these two sets, and through Equation 4.7, a two-layer classifier for the original sets* S_{rs} *and* $S^b_{r\bar{s}}$*.*

Algorithm 4.1 The outline of the two-stage construction algorithm of the classifier TLC^Q

Input:(i) $\widehat{P(\hat{S}_{rs})}$; (ii) $\widehat{P(\hat{S}^b_{r\bar{s}})}$.
Output:$TLC^Q = \langle \mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d} \rangle$.

1: Synthesize $(\mathbf{A}, \mathbf{b})$ such that Equation 4.25 is satisfied;
2: Compute the sets $I(\widehat{P(\hat{S}_{rs})})$ and $I(\widehat{P(\hat{S}^b_{r\bar{s}})})$ using $(\mathbf{A}, \mathbf{b})$;
3: Synthesize $(\mathbf{C}, \mathbf{d})$ to linearly separate $I(\widehat{P(\hat{S}_{rs})})$ and $I(\widehat{P(\hat{S}^b_{r\bar{s}})})$;
4: Return $(\mathbf{A}, \mathbf{b}, \mathbf{C}, \mathbf{d})$;

Proof: The result follows immediately from Lemma 4.2. □

To alleviate the computational complexity involved with the synthesis of the linear inequalities for each layer, we also consider the possibility of synthesizing these sets of linear inequalities in an incremental manner. In particular, for the first stage, we start by having all the pairs of safe and unsafe states in the set W, i.e., $W = \widehat{P(\hat{S}_{rs})} \times \widehat{P(\hat{S}^b_{r\bar{s}})}$. Next, we synthesize one linear inequality at a time; in particular, at each iteration, a linear inequality is synthesized to satisfy Equation 4.25 for the maximum number of pairs of safe and unsafe states from the set W, then these pairs are subsequently removed from W. The procedure is repeated until the set W is empty. For the second stage, we start by having the set $I_U = I(\widehat{P(\hat{S}^b_{r\bar{s}})})$. Next, we synthesize one linear inequality at at time; in particular, at each iteration, a linear inequality is synthesized to separate the maximum number of states of the set I_U from the states of the set $I(\widehat{P(\hat{S}_{rs})})$, and the separated states are removed from I_U. The procedure is repeated until the set I_U is empty.

In the rest of this subsection, first we give an MIP formulation that synthesizes the first-layer inequalities. Next, we give the additional, more incremental procedure to synthesize the first-layer inequalities described above. For the synthesis of the second layer-linear inequalities, using an MIP formulation and the incremental procedure that was described in the previous paragraph, the reader is referred to Sections 3.4 and 3.5.

4.2.4.1 *Synthesizing the inequalities of the first layer using Mathematical Programming*

In this part, we provide a MIP formulation for the construction of the first layer of the separator TLC^Q. The main input for this formulation is $\Upsilon \equiv \widehat{P(\hat{S}_{rs})} \cup \widehat{P(\hat{S}^b_{r\bar{s}})}$. Additionally, ϵ, M_1 and M_2 are analogous to their counterparts in Equations 4.8-4.22. Similarly, the variables α_i, $(\mathbf{A}[i,.], \mathbf{b}[i])$, and $\gamma_{\mathbf{x}}[i]$ are analogous to their counterparts in Equations 4.8-4.22. On the other hand, the binary variable $\theta_{\mathbf{su}}[i] = 1$ only if $\gamma_{\mathbf{s}}[i] = 0 \ \wedge \ \gamma_{\mathbf{u}}[i] = 1$. The formulation itself is expressed by the following equations:

$$\text{Min} \sum_{i=1}^{k_1} \alpha_i \tag{4.26}$$

$$\forall \mathbf{x} \in \{\Upsilon\}, \ \forall i \in \{1, \ldots, k_1\} : \ \epsilon + M_1 \cdot (\gamma_{\mathbf{x}}[i] - 1) \leq \mathbf{A}[i,.] \cdot \mathbf{x} - \mathbf{b}[i] \leq M_2 \cdot \gamma_{\mathbf{x}}[i] \tag{4.27}$$

$$\forall i \in \{1, \ldots, k_1\}, \ \forall \mathbf{s} \in \widehat{P(\hat{S}_{rs})}, \ \forall \mathbf{u} \in \widehat{P(\hat{S}^b_{r\bar{s}})} : \ 2 \cdot \theta_{\mathbf{su}}[i] \leq \gamma_{\mathbf{u}}[i] - \gamma_{\mathbf{s}}[i] + 1 \tag{4.28}$$

$$\forall \mathbf{s} \in \widehat{P(\hat{S}_{rs})}, \ \forall \mathbf{u} \in \widehat{P(\hat{S}^b_{r\bar{s}})} : \ \sum_{i=1}^{k_1} \theta_{\mathbf{su}}[i] \geq 1 \tag{4.29}$$

$$\forall i \in \{1, \ldots, k_1\}, \ \forall j \in \{1, \ldots, n\} : \ 0 \leq \mathbf{A}[i,j] \leq \alpha_i \tag{4.30}$$

$$\forall i \in \{1, \ldots, k_1\} : \ 0 \leq \mathbf{b}[i] \leq \alpha_i \tag{4.31}$$

$$\alpha_i, \ \gamma_{\mathbf{x}}[i], \ \theta_{\mathbf{su}}[i] \in \{0, 1\} \tag{4.32}$$

The objective of the formulation is defined as the minimization of the number of linear inequalities employed by the synthesized first layer. The mechanics of Equation 4.27 and Equations 4.30– 4.32 are very similar to the corresponding equations in the formulation given by Equations 4.8-4.22. On the other hand,

Equation 4.28 implies that $\theta_{\mathbf{su}}[i] = 1$ only if $\gamma_{\mathbf{u}}[i] = 1$ and $\gamma_{\mathbf{s}}[i] = 0$. In particular, $\gamma_{\mathbf{u}}[i] - \gamma_{\mathbf{s}}[i] + 1 \in \{0, 1, 2\}$. Thus, $\theta_{\mathbf{su}}[i]$ can be set to one only if $\gamma_{\mathbf{u}}[i] - \gamma_{\mathbf{s}}[i] + 1 = 2 \Leftrightarrow \gamma_{\mathbf{u}}[i] - \gamma_{\mathbf{s}}[i] = 1 \Leftrightarrow \gamma_{\mathbf{u}}[i] = 1 \ \wedge \ \gamma_{\mathbf{s}}[i] = 0$. Finally, Equation 4.29 implements the requirement of Equation 4.25. For a proper pricing of M_1 and M_2, an analysis similar to the one presented in Subsection 4.2.3 can be applied to get the conclusion that M_1 can be set equal to $1 + \epsilon$, and that M_2 can be set equal to ν.

Theorem 4.3 *The application of the formulation of Equations 4.26–4.32 to the sets* $\widehat{P(\hat{S}_{rs})}$ *and* $\widehat{P(\hat{S}^b_{r\bar{s}})}$ *corresponding to a given RAS* Φ*, returns a minimum-cardinality set of linear inequalities* $(\mathbf{A}, \mathbf{b})$ *that satisfies Equation 4.25.*

Complexity considerations The MIP formulation of Equations 4.26–4.32 involves $\mathcal{O}(m_u \cdot m_s \cdot k_1)$ binary variables, $\mathcal{O}(n \cdot k_1)$ real variables, and $\mathcal{O}(m_u \cdot m_s \cdot k_1 + k_1 \cdot n)$ technological constraints.

4.2.4.2 Iterative procedures for the first-layer construction

In this part, we present an iterative algorithm that synthesizes one linear inequality at each iteration. According to Lemma 4.2, the synthesized set of linear inequalities of the first layer should satisfy Equation 4.25. Therefore, we construct the set W that contains all the pairs of safe and unsafe states that have not yet been separated according to Equation 4.25 by any of the generated linear inequalities. At each iteration, a linear inequality is generated to separate the maximum number of pairs in W, and the separated pairs are removed from W. The algorithm terminates when W is empty, that is, Equation 4.25 is satisfied by the generated linear inequalities. Let $S \equiv \{\mathbf{s} \mid (\mathbf{s}, \mathbf{u}) \in W\}$ and $U \equiv \{\mathbf{u} \mid (\mathbf{s}, \mathbf{u}) \in W\}$ be the sets of safe and unsafe states in W. After each iteration, the sets S and U might shrink down, an effect that might result in having a dimension j such that $\forall \mathbf{s} \in S, \ \forall \mathbf{u} \in U : \ \mathbf{s}[j] \geq \mathbf{u}[j]$. Thus, this dimension can be dropped as explained in Subsection 4.2.2. To this end, we define J_0 as the set of dimensions removed from the state space by the projections

Algorithm 4.2 The iterative construction of the first layer of a two-layer classifier

Input:(i) $\widehat{P(\hat{S}_{rs})}$; (ii) $\widehat{P(\hat{S}^b_{r\bar{s}})}$; (iii) ϵ, M_1 and M_2
Output:$(\mathbf{A}, \mathbf{b})$

1: $W = \widehat{P(\hat{S}_{rs})} \times \widehat{P(\hat{S}^b_{r\bar{s}})}$; $\quad i = 0$;
2: $J_P = L_P$; $\;J_0 = \emptyset$;
3: **while** $W \neq \emptyset$ **do**
4: $\quad i = i + 1$;
5: $\quad$ **for all** $\{j \mid \forall(\mathbf{s}, \mathbf{u}) \in W : \; \mathbf{s}[j] \geq \mathbf{u}[j] \;\}$ **do**
6: $\quad\quad$ Remove j from J_P; Add j to J_0;
7: $\quad$ **end for**
8: $\quad W \leftarrow$ Project the elements of W to subspace V_{JP};
9: $\quad S = \{\mathbf{s} \mid (\mathbf{s}, \mathbf{u}) \in W\}$; $U = \{\mathbf{u} \mid (\mathbf{s}, \mathbf{u}) \in W\}$;
10: $\quad \Upsilon = S \cup U$;
11: $\quad$ Generate the linear inequality $(\mathbf{A}'[i,.], \mathbf{b}'[i])$ by solving the MIP formulation of Equations 4.34–4.38 with inputs W, Υ, ϵ, M_1 and M_2.
12: $\quad$ Construct $(\mathbf{A}[i,.], \mathbf{b}[i])$ from $(\mathbf{A}'[i,.], \mathbf{b}'[i])$ through Equation 4.33.
13: $\quad$ Remove the set $\{(\mathbf{s}, \mathbf{u}) \in W \mid \theta_{\mathbf{su}} = 1\}$ from W.
14: **end while**
15: Return $(\mathbf{A}, \mathbf{b})$;

applied in the course of this iterative procedure. Let $J_P \equiv L_P \setminus J_0$, and V_{JP} denote the $|J_P|$-dimensional subspace supporting the further projection. Finally, let $\Pi : \mathbb{N} \rightarrow \mathbb{N}$ be a bijection that maps the elements of the dimension set L_P to the dimensions of subspace V_{JP}. The linear inequalities in the subspace V_P are constructed from the linear inequalities synthesized in the projected subspace V_{JP} as follows:

$$\forall j \in J_0 : \; \mathbf{a}[j] = 0 \;\; \wedge \;\; \forall j \in J_P : \mathbf{a}[j] = \mathbf{a}'[\Pi(j)] \;\; \wedge \;\; \mathbf{b} = \mathbf{b}' \tag{4.33}$$

where $\langle \mathbf{A}', \mathbf{b}' \rangle$ are the linear inequalities synthesized in the projected subspace V_{JP}. The complete iterative procedure is depicted in Algorithm 4.2.

Next, we present the MIP formulation utilized by Algorithm 4.2 to synthesize a linear inequality that separates the maximum number of pairs of safe and unsafe states from W according to Equation 4.25. The main inputs to the formulation are: (i) the set W of pairs of safe and unsafe states that have not been yet separated according to Equation 4.25, and (ii) the set Υ of safe and unsafe states that appear in

W. Also, in the subsequent discussion, we set $w \equiv |W|$, $m_s \equiv |S|$, $m_u \equiv |U|$, and $n \equiv |J_P|$. Additionally, ϵ, M_1 and M_2 are similar to their counterparts in Equations 4.26-4.32. Finally, the variables $\mathbf{a}$, b, $\gamma_{\mathbf{x}}$, and $\theta_{\mathbf{su}}$ are analogous to their counterparts in Equations 4.26-4.32 with $k_1 = 1$. The formulation itself takes the following form:

$$\text{Max} \sum_{(\mathbf{s},\mathbf{u}) \in W} \theta_{\mathbf{su}} \tag{4.34}$$

$$\forall \mathbf{x} \in \{\Upsilon\} : \ \epsilon + M_1 \cdot (\gamma_{\mathbf{x}} - 1) \leq \mathbf{a} \cdot \mathbf{x} - b \leq M_2 \cdot \gamma_{\mathbf{x}} \tag{4.35}$$

$$\forall (\mathbf{s}, \mathbf{u}) \in W : \ 2 \cdot \theta_{\mathbf{su}} \leq \gamma_{\mathbf{u}} - \gamma_{\mathbf{s}} + 1 \tag{4.36}$$

$$\forall i \in \{1, \ldots, n\} : \ 0 \leq \mathbf{a}[i] \leq 1 \quad \wedge \quad 0 \leq b \leq 1 \tag{4.37}$$

$$\gamma_{\mathbf{x}}, \ \theta_{\mathbf{su}} \in \{0, 1\} \tag{4.38}$$

The objective of the formulation is to maximize the number of pairs of safe and unsafe states from the set W that are separated according to Equation 4.25. The mechanics of Equations 4.35–4.38 resemble their corresponding counterparts in the formulation given by Equations 4.26–4.32. Analyzing Equation 4.35, we can see that M_1 can be set equal to $1 + \epsilon$, and that M_2 can be set equal to ν.

Complexity considerations The MIP formulation of Equations 4.34–4.38 involves $\mathcal{O}(m_u + m_s + w)$ binary variables, $\mathcal{O}(n)$ real variables, and $\mathcal{O}(m_s + m_u + w)$ technological constraints.

Synthesizing an "upper-bound" linear inequality By an "upper-bound" linear inequality, we mean a linear inequality whose coefficients except for exactly one, are set to zero. Thus, it can be described as $\mathbf{a}[i^*] \cdot \mathbf{x}[i^*] \leq b$. The proof of Proposition 4.2 demonstrates the existence of a two-layer classifier whose first-layer

inequalities are upper-bound inequalities. A formulation that can provide these inequalities can be obtained through a straightforward modification of the formulation of Eqs 4.34–4.38. It should be clear that synthesizing an upper-bound linear inequality is computationally easier as it involves the selection of $\mathbf{a}[i^*]$ and b only rather than the selection of all the coefficients of the linear inequality $(\mathbf{a}, b)$. This effect will be demonstrated in the next subsection.

4.2.5 Computational results

In this subsection we report a set of experiments that demonstrates and assesses the applicability of the proposed methodologies for the construction of two-layer classifiers. First, we apply the introduced methods to the RAS configuration defined in Table 4.3 where the considered RAS has six resource types, $R_1, \ldots, R_6$, each with capacity $C_i = 3$. It also has four process types, $J_1, \ldots, J_4$, where each process type consists of a set of consecutive processing stages with each stage engaging a single resource type at the amount indicated by the corresponding coefficient; for instance, the first processing stage of the first process type engages two units of resource R_4, the second stage of the same type engages one unit of resource R_2, etc. The depicted RAS has 13 processing stages and this number defines the dimensionality of the state vector. The application of the DAP design methodology proposed in this section revealed that the considered RAS has a reachable state space of 19,980 states, with 13,092 of them being safe states and the remaining 6,888 being unsafe states. Applying the introduced thinning techniques led to a set $\widehat{P(\hat{S}_{rs})}$ with 6 states, and a set $\widehat{P(\hat{S}^b_{r\bar{s}})}$ with 3 states, whereas the dimensionality of the projected subspace V_P is 6. These sets are reported in Table 4.4. Furthermore, Table 4.4 reports the two-layer classifiers obtained by applying (i) Equations 4.8–4.22, (ii) Algorithm 4.1 in conjunction with Equations 4.26–4.32, (iii) Algorithm 4.1

Table 4.3: The RAS considered in the example of Subsection 4.2.5

Resource Types: $\{R_1, \ldots, R_6\}$ Resource Capacities: $C_i = 3, \forall i \in \{1, \ldots, 6\}$
Process Type 1: $2R_4 \rightarrow R_2 \rightarrow R_1$ Process Type 2: $R_3 \rightarrow 3R_6$ Process Type 3: $2R_1 \rightarrow R_5 \rightarrow 3R_1 \rightarrow R_6 \rightarrow 3R_2$ Process Type 4: $2R_3 \rightarrow R_1 \rightarrow 3R_5$

in conjunction with Algorithm 4.2, and (iv) Algorithm 4.1 in conjunction with Algorithm 4.2 and the restriction of the first-layer inequalities to the upper-bound type. The two-layer classifiers obtained by the first three methods have the same size which equals $(2 \cdot |L_P| + 1) \cdot k_1 + (2 \cdot k_1 + 1) \cdot k_2 = (2 \cdot 6 + 1) \cdot 3 + (2 \cdot 3 + 1) \cdot 1 = 46$. On the other hand, the size of the classifier obtained by the last method equals $(2 \cdot |L_P| + 1) \cdot k_1 + (2 \cdot k_1 + 1) \cdot k_2 = (2 \cdot 6 + 1) \cdot 7 + (2 \cdot 7 + 1) \cdot 2 = 121$.

Table 4.5 reports the results obtained from the application of the methods proposed in this section for the deployment of the maximally permissive DAP on additional 39 RAS configurations. These configurations provide a representative sample of the results obtained in our experiments. More specifically, for each of these configurations, Table 4.5 reports: (i) the total number of processing stages ξ, (ii) the cardinality of the reachable safe subspace S_{rs}, (iii) the cardinality of the reachable unsafe subspace $S_{r\bar{s}}$, (iv) the dimensionality of the projected subspace V_P, (v) the cardinality of the set of maximal projected reachable safe states $\widehat{P(\hat{S}_{rs})}$, (vi) the cardinality of the set of minimal projected boundary reachable unsafe states $\widehat{P(\hat{S}^b_{r\bar{s}})}$, (vii) the size of the two-layer classifier obtained by applying the formulation of Equations 4.8–4.22, (viii) the size of the two-layer classifier obtained by applying Algorithm 4.1 in conjunction with Equations 4.26–4.32, (ix) the size of the two-layer classifier obtained by applying Algorithm 4.1 in conjunction with Algorithm 4.2, and (x) the size of the two-layer classifier obtained by applying Algorithm 4.1 in conjunction with Algorithm 4.2 while

Table 4.4: The results of the application of the two-layer classifier design methods on the RAS configuration depicted in Table 4.3

$\widehat{P(\hat{S}_{rs})} = \{[3\ 3\ 0\ 0\ 0\ 3]^T,\ [1\ 3\ 0\ 0\ 1\ 3]^T,\ [0\ 3\ 1\ 2\ 0\ 0]^T,$ $[0\ 3\ 0\ 3\ 0\ 3]^T,\ [0\ 3\ 0\ 2\ 1\ 3]^T,\ [0\ 2\ 1\ 2\ 0\ 3]^T\}$ $\widehat{P(\hat{S}^b_{r\bar{s}})} = \{[0\ 3\ 1\ 0\ 0\ 1]^T,\ [0\ 0\ 0\ 3\ 1\ 0]^T,\ [1\ 0\ 0\ 1\ 0\ 0]^T\}$
The two-layer classifier obtained by the formulation of Equations 4.8–4.22
First layer inequalities : $\{\ 0.03 \cdot x_2 + 0.91 \cdot x_3 + 0.01 \cdot x_6 \leq 1;$ $x_1 + 0.01 \cdot x_4 + 0.01 \cdot x_5 \leq 0.03;$ $x_4 \leq 0\}$ Second layer inequalities: $\{\ \gamma_1 + 0.01 \cdot \gamma_2 + 0.01 \cdot \gamma_3 \leq 0.01\ \}$
The two-layer classifier obtained by Algorithm 4.1 in conjunction with Equations 4.26–4.32
First layer inequalities : $\{\ 0.03 \cdot x_2 + 0.03 \cdot x_3 + 0.01 \cdot x_6 \leq 0.12;$ $x_3 + x_4 \leq 0;$ $0.99 \cdot x_1 + 0.33 \cdot x_4 + 0.01 \cdot x_5 \leq 1\}$ Second layer inequalities: $\{\ \gamma_1 + \gamma_2 + \gamma_3 \leq 1\ \}$
The two-layer classifier obtained by Algorithm 4.1 in conjunction with Algorithm 4.2
First layer inequalities : $\{\ 0.99 \cdot x_1 + 0.33 \cdot x_4 + 0.01 \cdot x_5 \leq 1;$ $0.25 \cdot x_2 + 0.25 \cdot x_3 + 0.0833 \cdot x_6 \leq 1;$ $x_1 \leq 0\}$ Second layer inequalities: $\{\ 0.01 \cdot \gamma_1 + \gamma_2 + 0.01 \cdot \gamma_3 \leq 0.01\ \}$
The two-layer classifier obtained by Algorithm 4.1 in conjunction with Algorithm 4.2 and the restriction of the first-layer inequalities to the upper-bound type
First layer inequalities : $\{\ x_1 \leq 0.99;\ x_2 \leq 2.99;\ x_3 \leq 0.99;$ $x_4 \leq 2.99;\ x_4 \leq 0.99;\ x_5 \leq 0.99;$ $x_6 \leq 0.99\}$ Second layer inequalities: $\{\ 0.02 \cdot \gamma_1 + 0.01 \cdot \gamma_3 + 0.01 \cdot \gamma_4 +$ $0.02 \cdot \gamma_5 + 0.01 \cdot \gamma_6 \leq 0.03;$ $0.01 \cdot \gamma_2 + 0.01 \cdot \gamma_3 +$ $0.01 \cdot \gamma_6 \leq 0.02\ \}$

restricting the first layer to upper-bound linear inequalities. (xi) Finally, the last five columns (11-15) report, in secs, the computational times that were required for the generation of the corresponding classifiers in columns 7-10. In particular, column 11

reports the time that is required to perform the pre-processing steps for the generation of the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$, and it is common for all four approaches. The remaining four columns (12-15) report the computational times that are required in order to obtain the target classifier from the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$.

A "-" entry in the presented data indicates that the relevant methodology failed to find a solution within an 120 min time-budget or that it led to memory overflow.[3] On the other hand, the qualification [F] indicates that the obtained solution is just a feasible solution; this would happen if the solution algorithm was terminated prematurely, upon the exhaustion of the 120 min budget.

The following observations can be drawn from the reported results:

- Similar to the case of Chapter 3, the various "set-thinning" steps introduced in Section 2.5 play a very significant role in reducing the data to be considered explicitly in the synthesis of the sought classifiers, and therefore, in establishing the feasibility of the proposed methods.

- Restricting Algorithm 4.2 to consider only upper-bound linear inequalities for the first-layer, enables the solution of larger problem instances, and it tends to require significantly less time for the generation of the target classifier than the other approaches. On the other hand, this restriction typically increases the size of the synthesized classifier.

- Algorithm 4.2 is capable of solving larger problem instances compared to the approach that is based on Equations 4.26–4.32.

- For small problem instances, Equations 4.8–4.22 can be solved to optimality; hence, the minimum-size classifier is synthesized.

[3]Relaxing the time constraint did not help in finding a solution. For the sake of completeness, we also report that these experiments were performed on a 2.66 GHz quad-core Intel Xeon 5430 processor with 6 MB of cache memory and 32 GB RAM; however, each job ran on a single core.

Table 4.5: A sample of our experimental results for the construction of two-layer classifiers

$\|\xi\|$	$\|S_{rs}\|$	$\|S_{r\bar{s}}\|$	$\|L_P\|$	$\|\widehat{P(\hat{S}_{rs})}\|$	$\|\widehat{P(\hat{S}^b_{r\bar{s}})}\|$	$\|TLC_1\|$	$\|TLC_2\|$	$\|TLC_3\|$	$\|TLC_4\|$	t_g	t_1	t_2	t_3	t_4
15	5984	2560	8	22	15	86 [F]	77	107	302	0	7200	7200	0	0
15	3360	1728	7	11	8	52	52	69	192	0	901	11	0	0
12	3274	656	7	12	7	52 [F]	35	35	154	0	7200	10	0	0
9	3376	368	5	13	7	40	66	40	152	0	434	82	0	0
12	1270	682	8	4	15	39	39	58	134	0	8	1	0	0
12	892	461	7	7	6	52	59	52	213	0	843	22	0	0
12	1149	66	6	12	6	31	31	31	121	0	9	2	0	0
12	887	223	8	13	11	58	58	96	233	0	327	38	0	0
12	636	408	6	9	7	46	46	46	138	0	500	4	0	0
9	879	75	6	17	7	61 [F]	46	61	212	0	7200	45	0	0
9	753	159	6	8	6	46	46	61	138	0	83	2	0	0
10	728	56	5	23	12	27	27	27	131	0	33	29	0	0
26	196021	11451	8	46	15	-	170	86	275	4	-	7200	183	0
15	2827	1178	9	33	11	-	127	106	328	0	-	7200	1	0
12	354	259	9	20	10	-	127	147	274	0	-	7200	4	0
15	2030	1870	10	36	14	-	177	152	410	0	-	7200	98	0
64	34695	1773193	55	1612	427	-	-	11012	9180	54	-	-	2439	870
39	757699	700781	21	1927	89	-	-	2144	1603	54	-	-	2131	57
56	21099	906478	48	703	225	-	-	6164	6679	40	-	-	2698	375
64	16170	445499	51	1272	320	-	-	6012	7636	12	-	-	2471	388
56	8464	187610	45	622	250	-	-	3797	6113	7	-	-	776	275
28	99548	82989	19	747	75	-	-	1839	1965	4	-	-	316	63
78	3799	166858	70	549	407	-	-	5044	12254	7	-	-	1401	267
24	94431	72238	17	657	71	-	-	1228	1664	3	-	-	970	71
60	9448	152984	49	530	225	-	-	3597	6793	5	-	-	1097	305
24	104550	49620	15	1158	67	-	-	447	1089	2	-	-	1452	193
24	115766	28510	11	84	12	-	-	176	380	2	-	-	328	26
52	1622861	2600349	37	14257	257	-	-	-	4764	261	-	-	-	2143
52	853187	1993667	39	16036	576	-	-	-	7183	141	-	-	-	3199
63	65992	2105590	56	2203	708	-	-	-	12576	79	-	-	-	3594
65	344779	1230069	51	22876	905	-	-	-	12397	88	-	-	-	700
88	53080	1410311	74	5460	1046	-	-	-	17139	87	-	-	-	2467
64	118470	1253425	54	3203	612	-	-	-	10873	42	-	-	-	2024
56	55832	800208	48	3402	687	-	-	-	9046	31	-	-	-	2606
78	31856	740548	65	3077	933	-	-	-	14197	31	-	-	-	3542
48	80343	691959	39	2657	241	-	-	-	5471	26	-	-	-	634
54	36934	443053	48	1463	474	-	-	-	7885	15	-	-	-	848
91	6546	444303	82	998	684	-	-	-	17215	22	-	-	-	615
56	11803	413231	47	1874	755	-	-	-	8015	13	-	-	-	917

- Comparing the approach based on Equations 4.8–4.22 to the approach based on Algorithm 4.1 in conjunction with 4.26–4.32, we can see that the tractability of the second is slightly better than that of the first. Both methods are not scalable. On the other hand, the size of the synthesized classifiers are of comparable order.

4.3 Boolean classifiers

In this section, we show that for the considered RAS classes, the sought classifiers can also take the form of a Boolean function that is defined on top of a layer of linear inequalities. We also provide effective methods for constructing the proposed classifiers. Hence, this section evolves as follows: Subsection 4.3.1 provides a formal definition of the Boolean classification problem, and subsequently, it proceeds to show the completeness of the adopted classification structure w.r.t. the addressed RAS classes. In Subsection 4.3.2 the Boolean classification problem is reduced to an equivalent problem with a much smaller input set in terms of the explicitly considered state vectors and their dimensionality. Subsection 4.3.3 presents two types of Boolean classifiers with the relevant methodologies to synthesize these classifiers. Finally, Subsection 4.3.4 reports a series of experiments to demonstrate the proposed methodologies for the construction of Boolean classifiers, and assess their efficacy.

4.3.1 The Boolean classifier design problem

It has already been shown that, in the broader RAS class defined in Chapter 1, the convex hull of the safe states might contain unsafe states, and therefore, the safe subspace can not be characterized by a conjunction of linear inequalities. In other words, the safe subspace can not be characterized by a single polyhedron. However, in the following, we establish that the safe subspace can be characterized as the union of a set of polyhedra. From a more practical standpoint, this characterization is eventually materialized by a two-tier classification structure, where the constituent polyhedra are defined by a set of linear inequalities that constitutes the first layer of the classifier, while the union of these polyhedra is expressed by a Boolean function that constitutes its second layer. We shall proceed by first defining the geometric constructs employed by the Boolean classifier, and subsequently, we shall formally introduce this classifier and prove its capability of achieving the sought separation.

Definition 4.5 *1. A "*polyhedron*" P is a set that can be described in the form $\{\mathbf{x} \in \Re^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$, where $\mathbf{A}$ is an $m \times n$ real-valued matrix and $\mathbf{b}$ is a vector in $\Re^m$. In the following, P is defined by the 2-tuple $\langle \mathbf{A}, \mathbf{b} \rangle$.*

*2. A "*packing polyhedron*" is a polyhedron described in the form $\{\mathbf{x} \in \Re^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b},\ \mathbf{A} \geq \mathbf{0}$, *and* $\mathbf{b} \geq \mathbf{0}\}$ while a "*covering polyhderon*" is a polyhedron described in the form $\{\mathbf{x} \in \Re^n \mid \mathbf{A}\mathbf{x} \geq \mathbf{b},\ \mathbf{A} \geq \mathbf{0}$, *and* $\mathbf{b} \geq \mathbf{0}\}$.*

Next, we introduce the following indicators to characterize the membership of a point in a polyhedron.

Definition 4.6 *Given a point $\mathbf{x}$, a packing polyhedron $\Lambda_{i'} = \langle \mathbf{A}_{i'}, \mathbf{b}_{i'} \rangle$, and a covering polyhedron $\Psi_{i^*} = \langle \mathbf{A}_{i^*}, \mathbf{b}_{i^*} \rangle$:*

*1. A "*Packing Inclusion Indicator*" $\chi(\mathbf{x}, \Lambda_{i'}) = 1$ iff $\mathbf{A}_{i'}\ \mathbf{x} \leq \mathbf{b}_{i'}$.*

*2. A "*Covering Exclusion Indicator*" $\kappa(\mathbf{x}, \Psi_{i^*}) = 0$ iff $\mathbf{A}_{i^*}\ \mathbf{x} \geq \mathbf{b}_{i^*}$.*

The next corollary is an immediate consequence of the monotonicity property of Proposition 2.1.

Corollary 4.1 *Consider a packing polyhedron $\Lambda_{i'} = \{\mathbf{x} \in \Re^\xi \mid \mathbf{x} \leq \mathbf{s}'\}$ and a covering polyhedron $\Psi_{i^*} = \{\mathbf{x} \in \Re^\xi \mid \mathbf{x} \geq \mathbf{u}'\}$ where $\mathbf{s}'$ is a safe state and $\mathbf{u}'$ is an unsafe state. Then, we have:*

(i) $\forall \mathbf{u} \in S_{r\bar{s}} : \chi(\mathbf{u}, \Lambda_{i'}) = 0$; *(ii)* $\forall \mathbf{s} \in S_{rs} : \kappa(\mathbf{s}, \Psi_{i^*}) = 1$.

Next, we introduce the Boolean classifier.

Definition 4.7 *Consider two vector sets G and H from an ξ-dimensional vector space V. A Boolean classifier BC is defined as a 3-tuple $\langle \Lambda, \Psi, F \rangle$, where (i) $\Lambda = \{\Lambda_1, \ldots, \Lambda_\alpha\}$ is a set of packing polyhedra; (ii) $\Psi = \{\Psi_1, \ldots, \Psi_\beta\}$ is a set of covering polyhedra; (iii) $F : \{0,1\}^{(\alpha+\beta)} \rightarrow \{0,1\}$ is a Boolean function such that: $F(\chi(\mathbf{x}, \Lambda_1), \ldots,$*

$\chi(\mathbf{x},\Lambda_\alpha),\kappa(\mathbf{x},\Psi_1),\ldots,\kappa(\mathbf{x},\Psi_\beta))=1$ iff $\mathbf{x}\in G$. *Moreover, F is restricted to have no negated variables, i.e., it is composed of conjunction and disjunction operations only. The size of the Boolean classifier $|BC|$ is determined by the total number of operations required to classify a given vector, i.e.,* $(2\xi+1)\cdot(\sum_{i=1}^{\alpha} m_{\Lambda_i}+\sum_{i=1}^{\beta} m_{\Psi_i})+|F|$.

The expression that defines the size $|BC|$ of the Boolean classifier is explained as follows: the term $(\sum_{i=1}^{\alpha} m_{\Lambda_i}+\sum_{i=1}^{\beta} m_{\Psi_i})$ represents the total number of linear inequalities in the classifier. To evaluate each linear inequality, ξ multiplication operations, ξ addition operations, and one comparison operation are required. On the other hand, the term $|F|$ denotes the size of the Boolean function F, and it is defined by the total number of conjunctions and disjunctions that are involved in the definition of this function.

In the following, we shall index the polyhedra of the Boolean classifier as follows: $\Lambda_i=\langle\mathbf{A}_i,\mathbf{b}_i\rangle$, $i=1:\alpha$, and $\Psi_i=\langle\mathbf{A}_{\alpha+i},\mathbf{b}_{\alpha+i}\rangle$, $i=1:\beta$. Furthermore, in order to simplify the notation, we shall refer to $\chi(\mathbf{x},\Lambda_i)$ as χ_i and to $\kappa(\mathbf{x},\Psi_i)$ as κ_i. An example for a valid F is

$$F(\chi(\mathbf{x},\Lambda_1),\chi(\mathbf{x},\Lambda_2),\kappa(\mathbf{x},\Psi_1),\kappa(\mathbf{x},\Psi_2))=(\chi_1\wedge\kappa_1)\vee(\chi_2\wedge\kappa_2)$$

On the other hand, an example for an invalid F is

$$F(\chi(\mathbf{x},\Lambda_1),\chi(\mathbf{x},\Lambda_2),\kappa(\mathbf{x},\Psi_1),\kappa(\mathbf{x},\Psi_2))=(\chi_1\wedge\kappa_1)\vee(\chi_2\wedge\bar{\kappa_2})$$

The latter function is invalid because it has a negated variable. Based on the aforementioned classifier definition, the problem addressed in this section can be succinctly stated as follows:

Definition 4.8 – ***The Boolean classification problem:*** *Given a RAS Φ, construct a minimum-sized Boolean classifier for the vector sets corresponding to the subspaces S_{rs} and $S_{r\bar{s}}$, i.e., the reachable safe and the reachable unsafe states of the considered RAS Φ.*

The next proposition plays a central role in this section.

Proposition 4.6 *Given the two sets S_{rs} and $S_{r\bar{s}}$, there exists a Boolean classifier that separates the two subspaces.*

Proof: Assume that $S_{rs} = \{\mathbf{s}_1, \ldots, \mathbf{s}_\alpha\}$. Define the packing polyhedron $\Lambda_i = \{\mathbf{x} \in \Re^\xi \mid \mathbf{x} \leq \mathbf{s}_i\}$. Hence, $\forall i : \chi(\mathbf{s}_i, \Lambda_i) = 1$. An immediate result from Corollary 4.1 is that $\forall \mathbf{u} \in S_{r\bar{s}},\ \forall i : \chi(\mathbf{u}, \Lambda_i) = 0$. Setting $\Lambda = \bigcup_i \Lambda_i,\ \Psi = \emptyset$, and $F = \bigvee_{i \in \{1,\ldots,\alpha\}} \chi_i$, it follows that $\forall i \in \{1, \ldots, \alpha\} : F(\chi(\mathbf{s}_i, \Lambda_1), \ldots, \chi(\mathbf{s}_i, \Lambda_\alpha)) = 1$, and $\forall \mathbf{u} \in S_{r\bar{s}} : F(\chi(\mathbf{u}, \Lambda_1), \ldots, \chi(\mathbf{u}, \Lambda_\alpha)) = 0$. Therefore, the Boolean classifier $BC = \langle \Lambda, \Psi, F \rangle$ establishes the sought separation. □

4.3.2 Simplifying the Boolean classification problem

A major difficulty for the systematic construction of the Boolean classifier for the RAS encountered in various practical application contexts, is the huge cardinality of the sets S_{rs} and $S_{r\bar{s}}$. Thus, we utilize the thinning techniques developed in Section 2.5 to alleviate the computational complexity. Next, we establish the validity of those thinning operations in the context of this classification problem. The presented results are similar, in spirit, to their counterparts in Chapter 3, although the relevant proofs differ in their technicalities. We start with the following lemma:

Lemma 4.4 *Consider the Boolean classifiers $BC = \langle \Lambda, \Psi, F \rangle$ and $BC' = \langle \Lambda', \Psi', F \rangle$ where: (i) $|\Lambda| = |\Lambda'| = \alpha$, (ii) $|\Psi| = |\Psi'| = \beta$, (iii) $m_{\Lambda_i} = m_{\Lambda'_i},\ i = 1 : \alpha$, and (iv) $m_{\Psi_i} = m_{\Psi'_i},\ i = 1 : \beta$. Let $\langle \mathbf{A}_i, \mathbf{b}_i \rangle,\ i = 1 : \alpha + \beta$, be the polyhedra of BC, and let $\langle \mathbf{A'}_i, \mathbf{b'}_i \rangle,\ i = 1 : \alpha + \beta$, be the polyhedra of BC'. Then, for any pair of vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, the following holds true:*

- *If $\mathbf{x}_1$ is classified as a safe state by BC, and $\mathbf{A'}_i\ \mathbf{x}_2 - \mathbf{b'}_i \leq \mathbf{A}_i\ \mathbf{x}_1 - \mathbf{b}_i,\ i = 1 : \alpha + \beta$, then $\mathbf{x}_2$ is also classified as a safe state by BC'.*

- *If* $\mathbf{x}_1$ *is classified as an unsafe state by BC, and* $\mathbf{A'}_i\ \mathbf{x}_2 - \mathbf{b'}_i \geq \mathbf{A}_i\ \mathbf{x}_1 - \mathbf{b}_i,\ i = 1 : \alpha + \beta$*, then* $\mathbf{x}_2$ *is also classified as an unsafe state by BC′.*

Proof: Assume that $\mathbf{x}_1$ is classified as a safe state by BC. Thus,

$$F(\chi(\mathbf{x}_1, \Lambda_1), \ldots, \chi(\mathbf{x}_1, \Lambda_\alpha), \kappa(\mathbf{x}_1, \Psi_1), \ldots, \kappa(\mathbf{x}_1, \Psi_\beta)) = 1$$

Assume also that $\mathbf{A'}_i\ \mathbf{x}_2 - \mathbf{b'}_i \leq \mathbf{A}_i\ \mathbf{x}_1 - \mathbf{b}_i,\ i = 1 : \alpha + \beta$. Hence, Definitions 4.5 and 4.6 imply that if $\chi(\mathbf{x}_1, \Lambda_i) = 1$, then $\chi(\mathbf{x}_2, \Lambda'_i) = 1$, and that, if $\kappa(\mathbf{x}_1, \Psi_i) = 1$, then $\kappa(\mathbf{x}_2, \Psi'_i) = 1$. Therefore, all the "*On*" variables in $F(\chi(\mathbf{x}_1, \Lambda_1), \ldots, \chi(\mathbf{x}_1, \Lambda_\alpha), \kappa(\mathbf{x}_1, \Psi_1), \ldots, \kappa(\mathbf{x}_1, \Psi_\beta))$ are also "*On*" in $F(\chi(\mathbf{x}_2, \Lambda'_1), \ldots, \chi(\mathbf{x}_2, \Lambda'_\alpha), \kappa(\mathbf{x}_2, \Psi'_1), \ldots, \kappa(\mathbf{x}_2, \Psi'_\beta))$. Since F has no negated variables, then

$$F(\chi(\mathbf{x}_2, \Lambda'_1), \ldots, \chi(\mathbf{x}_2, \Lambda'_\alpha), \kappa(\mathbf{x}_2, \Psi'_1), \ldots, \kappa(\mathbf{x}_2, \Psi'_\beta)) = 1$$

Therefore, $\mathbf{x}_2$ is classified as a safe state by BC'.

Next, we prove the second result. Assume that $\mathbf{x}_1$ is classified as an unsafe state by BC. Thus,

$$F(\chi(\mathbf{x}_1, \Lambda_1), \ldots, \chi(\mathbf{x}_1, \Lambda_\alpha), \kappa(\mathbf{x}_1, \Psi_1), \ldots, \kappa(\mathbf{x}_1, \Psi_\beta)) = 0$$

Assume also that $\mathbf{A'}_i\ \mathbf{x}_2 - \mathbf{b'}_i \geq \mathbf{A}_i\ \mathbf{x}_1 - \mathbf{b}_i,\ i = 1 : \alpha + \beta$. Therefore, Definitions 4.5 and 4.6 imply that if $\chi(\mathbf{x}_1, \Lambda_i) = 0$, then $\chi(\mathbf{x}_2, \Lambda'_i) = 0$, and that if $\kappa(\mathbf{x}_1, \Psi_i) = 0$, then $\kappa(\mathbf{x}_2, \Psi'_i) = 0$. Therefore, all the "*Off*" variables in $F(\chi(\mathbf{x}_1, \Lambda_1), \ldots, \chi(\mathbf{x}_1, \Lambda_\alpha), \kappa(\mathbf{x}_1, \Psi_1), \ldots, \kappa(\mathbf{x}_1, \Psi_\beta))$ are also "*Off*" in $F(\chi(\mathbf{x}_2, \Lambda'_1), \ldots, \chi(\mathbf{x}_2, \Lambda'_\alpha), \kappa(\mathbf{x}_2, \Psi'_1), \ldots, \kappa(\mathbf{x}_2, \Psi'_\beta))$. Since F has no negated variables, then

$$F(\chi(\mathbf{x}_2, \Lambda'_1), \ldots, \chi(\mathbf{x}_2, \Lambda'_\alpha), \kappa(\mathbf{x}_2, \Psi'_1), \ldots, \kappa(\mathbf{x}_2, \Psi'_\beta)) = 0$$

Therefore, $\mathbf{x}_2$ is classified as an unsafe state by BC'.□

Thinning the sets S_{rs} and $S^b_{r\bar{s}}$ by respectively focusing on their maximal and minimal elements It is possible to obtain a Boolean classifier for the entire sets

S_{rs} and $S^b_{r\bar{s}}$, by focusing the classifier design process only on the maximal elements of the first set, $\hat{S}_{rs}$, and the minimal elements of the second set, $\hat{S}^b_{r\bar{s}}$.

Proposition 4.7 *Any Boolean classifier $BC = \langle \Lambda, \Psi, F \rangle$ that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ is also an effective separator for the entire sets S_{rs} and $S^b_{r\bar{s}}$.*

Proof: Let $\mathbf{s} \in S_{rs}$ be an arbitrary non-maximal reachable safe state vector, and $\mathbf{s}^* \in \hat{S}_{rs}$ be a maximal reachable safe state vector such that $\mathbf{s}^* \succ \mathbf{s}$.

Also, let $\mathbf{u} \in S^b_{r\bar{s}}$ be an arbitrary non-minimal boundary reachable unsafe state vector, and $\mathbf{u}^* \in \hat{S}^b_{r\bar{s}}$ be a minimal boundary reachable unsafe state vector such that $\mathbf{u}^* \prec \mathbf{u}$.

Assume that we have already identified a Boolean classifier $BC = \langle \Lambda, \Psi, F \rangle$ that separates $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$. Then, the non-negativity of all the coefficients of the linear inequalities of all the polyhedra, combined with the presumed relations of $\mathbf{s}$ to $\mathbf{s}^*$ and of $\mathbf{u}$ to $\mathbf{u}^*$, further imply that:

- $\mathbf{A}_i\ \mathbf{s} - \mathbf{b}_i \leq \mathbf{A}_i\ \mathbf{s}^* - \mathbf{b}_i,\ i = 1 : \alpha + \beta$. Applying Lemma 4.4, we can infer that $\mathbf{s}$ is classified as a safe state.

- $\mathbf{A}_i\ \mathbf{u} - \mathbf{b}_i \geq \mathbf{A}_i\ \mathbf{u}^* - \mathbf{b}_i,\ i = 1 : \alpha + \beta$. Applying Lemma 4.4, we can infer that $\mathbf{u}$ is classified as an unsafe state.

Since states $\mathbf{s}$ and $\mathbf{u}$ were arbitrarily chosen, we can infer that the Boolean classifier $BC = \langle \Lambda, \Psi, F \rangle$ is also an effective separator for the entire sets S_{rs} and $S^b_{r\bar{s}}$. □

Converting the separation problem of $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ to an equivalent separation problem of reduced dimensionality As explained in Section 2.5, after thinning the sets of safe states and boundary unsafe states to their respective maximal and minimal elements, we have frequently encountered a situation where many components of the vectors included in the set $\hat{S}_{rs}$ are always greater than or equal to the corresponding components of the vectors included in the set $\hat{S}^b_{r\bar{s}}$. We shall

see in the next proposition that dropping these dimensions does not compromise the feasibility or the optimality of the Boolean classification problem. In the statement and the proof of this proposition, we employ the relevant notation and terminology that were introduced in Section 2.5.

Proposition 4.8 *There exists a Boolean classifier BC^Q in the reduced subspace that separates the projected sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$ in subspace V_P,* iff *there exists a Boolean classifier BC^H, that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ in the original space V.*

Proof: First we show that the existence of a Boolean classifier $BC^Q = \left\langle \Lambda^Q, \Psi^Q, F \right\rangle$ for the projected sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$ in subspace V_P, implies the existence of a Boolean classifier $BC^H = \left\langle \Lambda^H, \Psi^H, F \right\rangle$ that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ in the original space V. Construct Λ^H and Ψ^H as follows:

$$\forall i, \forall l, \forall j \in L_0 : \mathbf{A}^H_i[l,j] = 0 \wedge \forall i, \forall l, \forall j \in L_P : \mathbf{A}^H_i[l,j] = \mathbf{A}^Q_i[l, \Pi(j)] \wedge \mathbf{b}^H_i = \mathbf{b}^Q_i \quad (4.39)$$

Then, we can see that

$$\forall \mathbf{x} \in \hat{S}_{rs} \cup \hat{S}^b_{r\bar{s}}, \forall i, \forall l : \mathbf{A}^H_i[l,.] \cdot \mathbf{x} = \sum_{j \in L} \mathbf{A}^H_i[l,j] \cdot \mathbf{x}[j] = \sum_{j \in L_P} \mathbf{A}^H_i[l,j] \cdot \mathbf{x}[j] = \mathbf{A}^Q_i[l,.] \cdot \mathbf{x}^p \quad (4.40)$$

where $\mathbf{x}^p$ is the image of $\mathbf{x}$ in subspace V_P. Using Definitions 4.5– 4.6 and Equation 4.40, we can infer that $\forall i : \chi(\mathbf{x}, \Lambda^H_i) = \chi(\mathbf{x}^p, \Lambda^Q_i)$, and that $\forall j : \kappa(\mathbf{x}, \Psi^H_j) = \kappa(\mathbf{x}^p, \Psi^Q_j)$. Therefore, $\mathbf{x}$ will have the same classification as $\mathbf{x}^p$. Therefore, $BC^H = \left\langle \Lambda^H, \Psi^H, F \right\rangle$ separates $\hat{S}_{rs}$ from $\hat{S}^b_{r\bar{s}}$ and the forward part is proved.

Next, we show the validity of the reverse part of the proposition, i.e., that the existence of a Boolean classifier $BC^H = \left\langle \Lambda^H, \Psi^H, F \right\rangle$ that separates the sets $\hat{S}_{rs}$ and $\hat{S}^b_{r\bar{s}}$ in the original space V, implies the existence of a Boolean classifier $BC^Q = \left\langle \Lambda^Q, \Psi^Q, F \right\rangle$ that separates the projected sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$ in subspace V_P. The rest of the proof proceeds similarly to the proof provided for the forward part of

the proposition, but it also relies on the fact that

$$\forall \mathbf{s} \in \hat{S}_{rs},\ \forall \mathbf{u} \in \hat{S}^b_{r\bar{s}},\ \forall j \in L_0 :\ \mathbf{s}[j] \geq \mathbf{u}[j] \tag{4.41}$$

Construct Λ^Q and Ψ^Q as follows:

$$\forall i, \forall l, \forall j \in L_P : \mathbf{A}^Q_i[l, \Pi(j)] = \mathbf{A}^H_i[l, j] \wedge \mathbf{b}^Q_i = \mathbf{b}^H_i - min_{\mathbf{s} \in \hat{S}_{rs}} \{ \sum_{j \in L_0} \mathbf{A}^H_i[l, j] \cdot \mathbf{s}[j] \} \tag{4.42}$$

Let $\mathbf{x} \in \hat{S}_{rs}$ and $\mathbf{x}^p \in P(\hat{S}_{rs})$ be the image of $\mathbf{x}$ in subspace V_P. We can see that whenever $\mathbf{A}^H_i[l, .] \cdot \mathbf{x} \leq \mathbf{b}^H_i[l]$, we have

$$\mathbf{A}^Q_i[l, .] \cdot \mathbf{x}^p = \sum_{j \in L_p} \mathbf{A}^H_i[l, j] \cdot \mathbf{x}[j] \leq \mathbf{b}^H_i[l] - \sum_{j \in L_0} \mathbf{A}^H_i[l, j] \cdot \mathbf{x}[j] \leq$$
$$\mathbf{b}^H_i[l] - min_{\mathbf{s} \in \hat{S}_{rs}} \{ \sum_{j \in L_0} \mathbf{A}^H_i[l, j] \cdot \mathbf{s}[j] \} = \mathbf{b}^Q_i$$

Applying Lemma 4.4 on $\mathbf{x}$, $\mathbf{x}^p$, BC^H, and BC^Q, we can infer that $\mathbf{x}^p$ is classified as a safe state by BC^Q.

Similarly, let $\mathbf{y} \in \hat{S}^b_{r\bar{s}}$ and $\mathbf{y}^p \in P(\hat{S}^b_{r\bar{s}})$ be the image of $\mathbf{y}$ in subspace V_P. We can see that whenever $\mathbf{A}^H_i[l, .] \cdot \mathbf{y} \geq \mathbf{b}^H_i[l]$, we have

$$\forall i, \forall l : \mathbf{A}^Q_i[l, .] \cdot \mathbf{y}^p = \sum_{j \in L_p} \mathbf{A}^H_i[l, j] \cdot \mathbf{y}[j] \geq \mathbf{b}^H_i[l] - \sum_{j \in L_0} \mathbf{A}^H_i[l, j] \cdot \mathbf{y}[j] \geq$$
$$\mathbf{b}^H_i[l] - max_{\mathbf{u} \in \hat{S}^b_{r\bar{s}}} \{ \sum_{j \in L_0} \mathbf{A}^H_i[l, j] \cdot \mathbf{u}[j] \} \geq$$
$$\mathbf{b}^H_i[l] - min_{\mathbf{s} \in \hat{S}_{rs}} \{ \sum_{j \in L_0} \mathbf{A}^H_i[l, j] \cdot \mathbf{s}[j] \} = \mathbf{b}^Q_i$$

where the last inequality is an implication of Equation 4.41 and the non-negativity of the elements of $\mathbf{A}^H_i$. Applying Lemma 4.4 on $\mathbf{y}$, $\mathbf{y}^p$, BC^H, and BC^Q, we can infer that $\mathbf{y}^p$ is classified as an unsafe state by BC^Q. Therefore, $BC^Q = \left\langle \Lambda^Q, \Psi^Q, F \right\rangle$ separates $P(\hat{S}_{rs})$ from $P(\hat{S}^b_{r\bar{s}})$. □

The practical implication of Proposition 4.8 is that we can construct a Boolean classifier BC^H for the sets $\hat{S}^b_{r\bar{s}}$ and $\hat{S}_{rs}$ by first developing a Boolean classifier BC^Q

for the projected sets $P(\hat{S}^b_{r\bar{s}})$ and $P(\hat{S}_{rs})$, and subsequently constructing BC^H from BC^Q through Equation 4.39, which is repeated here for emphasis and convenience:

$$\begin{aligned} &\forall i, \forall l, \forall j \in L_0 : \mathbf{A}^H_i[l,j] = 0 \ \wedge \\ \forall i, \forall l, \forall j \in L_P : \mathbf{A}^H_i[l,j] &= \mathbf{A}^Q_i(l, \Pi(j)) \wedge \mathbf{b}^H_i = \mathbf{b}^Q_i \end{aligned} \tag{4.43}$$

Furthermore, the second part of the proof of Proposition 4.8 guarantees the we can have a Boolean separator BC^Q for the projected sets $P(\hat{S}^b_{r\bar{s}})$ from $P(\hat{S}_{rs})$.

Some further simplifications As explained in Section 2.5, the projection P might introduce some redundancy and dominance among the elements of $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$. Hence, these effects are identified, and subsequently, removed from each set, resulting in the generation of the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$. The fact that this additional thinning does not compromise the effectiveness of the obtained separator BC^Q with respect to the separation of the sets $P(\hat{S}_{rs})$ and $P(\hat{S}^b_{r\bar{s}})$, can be argued using the same logic of Proposition 4.7.

4.3.3 Synthesizing the Boolean classifier BC^Q

In the light of the positioning of the classifier synthesis problem given in the first part of the section, the problem reduces to finding a set of polyhedra and a Boolean function that together can effect the sought dichotomy in the subspace V_P. In the approach pursued in this section, the Boolean function is set a priori; hence, the problem reduces to finding a set of polyhedra that, in collaboration with the given Boolean function, effects the sought dichotomy. The Boolean function itself is the major differentiator between the subclasses of Boolean classifiers that will be introduced in this section. In particular, we restrict our search to two subclasses of Boolean classifiers: "*Simple Boolean classifiers*" and "*Alternating Boolean classifiers*". In this subsection, we shall define each subclass and present the relevant methodology to synthesize it.

4.3.3.1 Simple Boolean classifier

Definition 4.9 *A* Simple Boolean classifier*,* BC_S^Q*, is a Boolean classifier such that* $\Lambda = \{\Lambda_1, \ldots, \Lambda_\alpha\}$*,* $\Psi = \emptyset$*, and* $F = \bigvee_{i=1}^{\alpha} \chi_i$*. Thus,* $|BC_S^Q| = (2|L_P|+1)\cdot(\sum_{i=1}^{\alpha} m_{\Lambda_i})+\alpha$*.*

Proposition 4.9 *Given the two sets* $\widehat{P(\hat{S}_{rs})}$ *and* $\widehat{P(\hat{S}^b_{r\bar{s}})}$*, there exists a Simple Boolean classifier that separates the two subspaces.*

Proof: Following the same logic as in the proof of Proposition 4.6, we assume that $S_{rs} = \{\mathbf{s}_1, \ldots, \mathbf{s}_\alpha\}$. Then, defining the packing polyhedra $\Lambda_i = \{\mathbf{x} \in \Re^\xi \mid \mathbf{x} \leq \mathbf{s}_i\}$, $i = 1:\alpha$, the sought separation is established by the Boolean classifier $BC =< \cup_i \Lambda_i, \emptyset, \bigvee_{i\in\{1,\ldots,\alpha\}} \chi i >$. $\square$

Since $\Psi = \emptyset$ and F is already defined, constructing a *parsimonious Simple Boolean classifier* reduces to synthesizing a set of packing polyhedra that effects the dichotomy and minimizes the size of the classifier. Two methodologies are given to support the computation of the aforementioned set of polyhedra. The first synthesizes a Simple Boolean classifier of minimal size through solving a Mixed Integer Programming formulation. The second is an iterative algorithm supported by a simpler Mixed Integer Programming formulation.

Synthesizing a minimal Simple Boolean classifier using Mathematical Programming We provide a MIP formulation that synthesizes a set of packing polyhedra that defines the sought simple classifier. In addition to $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$, the following parameters are used to provide additional control to the MIP formulation: (i) Parameters w and k that provide upper bounds for the number of inequalities employed by a single polyhedron and the number of the packing polyhedra employed by the sought separator respectively. These upper bounds are provided by $|L_P|$ and $|\widehat{P(\hat{S}_{rs})}|$ respectively. (ii) Strictly positive parameters ϵ, M_1 and M_2. In the subsequent discussion, let $\Upsilon \equiv \widehat{P(\hat{S}_{rs})} \cup \widehat{P(\hat{S}^b_{r\bar{s}})}$, and $n \equiv |L_P|$. The variables employed by the proposed formulation are as follows:

- z_i, $i = 1, \ldots, w * k$, is a binary variable set to one *iff* the i-th linear inequality is used for separation.

- θ_l, $l = 1, \ldots, k$, is a binary variable set to one *iff* the l-th polyhedron is used for separation.

- $(\mathbf{A}[i,.], \mathbf{b}[i])$, $i = 1, \ldots, w * k$, are the coefficients to be employed by the i-th linear inequality.

- $\delta_{\mathbf{x}}[i]$, $\mathbf{x} \in \Upsilon$, $i = 1, \ldots, w * k$, is a binary variable set to one *iff* $\mathbf{x}$ violates the linear inequality $\mathbf{A}[i,.] \cdot \mathbf{x} \leq \mathbf{b}[i]$.

- $\Delta_{\mathbf{x}}[l]$, $\mathbf{x} \in \Upsilon$, $l = 1, \ldots, k$, is a binary variable set to one *iff* $\mathbf{x}$ lies inside the l-th polyhedron.

- $r_{\mathbf{x}}[l]$, $\mathbf{x} \in \Upsilon$, $l = 1, \ldots, k$, is an integer variable set to zero *iff* $\mathbf{x}$ lies inside the l-th polyhedron.

Finally, the formulation itself takes the following form:

$$\text{Min} \sum_{l=1}^{k} \theta_l + (2n+1) \cdot (\sum_{i=1}^{k*w} z_i) \tag{4.44}$$

$$\forall \mathbf{x} \in \Upsilon, \forall i \in \{1, \ldots, k * w\} : \ \epsilon + M_1 \cdot (\delta_{\mathbf{x}}[i] - 1) \leq \mathbf{A}[i,.] \cdot \mathbf{x} - \mathbf{b}[i] \leq M_2 \cdot \delta_{\mathbf{x}}[i] \tag{4.45}$$

$$\forall \mathbf{x} \in \Upsilon, \forall l \in \{1, \ldots, k\} : \ \sum_{i=1}^{w} \delta_{\mathbf{x}}[l * w + i] + r_{\mathbf{x}}[l] = 0 \tag{4.46}$$

$$\forall \mathbf{x} \in \Upsilon, \forall l \in \{1, \ldots, k\} : \ w * (\Delta_{\mathbf{x}}[l] - \theta_l) \leq r_{\mathbf{x}}[l] \leq \Delta_{\mathbf{x}}[l] - \theta_l \tag{4.47}$$

$$\forall \mathbf{s} \in \widehat{P(\hat{S}_{rs})} : \ \sum_{l=1}^{k} \Delta_{\mathbf{s}}[l] \geq 1 \tag{4.48}$$

$$\forall \mathbf{u} \in \widehat{P(\hat{S}^{b}_{r\bar{s}})} : \ \sum_{l=1}^{k} \Delta_{\mathbf{u}}[l] = 0 \tag{4.49}$$

$$\forall i \in \{1,\ldots,k*w\}, \forall j \in \{1,\ldots,n\} :\ 0 \leq \mathbf{A}[i,j] \leq z_i \tag{4.50}$$

$$\forall i \in \{1,\ldots,k*w\} :\ 0 \leq \mathbf{b}[i] \leq z_i \tag{4.51}$$

$$\forall l \in \{1,\ldots,k\}, \forall i \in \{1,\ldots,w\} :\ z_{l*w+i} \leq \theta_l \tag{4.52}$$

$$z_i,\ \theta_l,\ \delta_{\mathbf{x}}[i],\ \Delta_{\mathbf{x}}[l] \in \{0,1\} \tag{4.53}$$

The validity of the above MIP formulation as a construction tool for the sought classifier BC_S^Q can be established as follows: First, the reader should notice that Equation 4.44 defines the objective of the formulation as the minimization of the size of the classifier. The separation logic is primarily expressed by the constraints of Equations 4.45–4.49. More specifically, the constraints of Equation 4.45 force $\delta_{\mathbf{x}}[i]$ to zero if $\mathbf{A}[i,.] \cdot \mathbf{x} \leq \mathbf{b}[i]$, and to one if $\mathbf{A}[i,.] \cdot \mathbf{x} > \mathbf{b}[i]$. The constraints of Equation 4.46 implement the conjunction of the linear inequalities that constitute the l-th polyhedron. To understand the mechanics of Equation 4.46, assume that the state vector $\mathbf{x}$ lies inside the l-th polyhedron. Then, $\mathbf{x}$ lies below all the linear inequalities defining the l-th polyhedron. Therefore, $\forall i \in \{1,\ldots,w\} : \delta_{\mathbf{x}}[l*w+i] = 0$, and consequently $r_{\mathbf{x}}[l] = 0$. On the other hand, assume that the state vector $\mathbf{x}$ lies outside the l-th polyhedron. Then $\mathbf{x}$ violates at least one of the linear inequalities defining the l-th polyhedron. Therefore, $\exists i \in \{1,\ldots,w\}$ s.t. $\delta_{\mathbf{x}}[l*w+i] = 1$, and consequently $r_{\mathbf{x}}[l] < 0$.

For a polyhedron l that is employed by the classifier ($\theta_l = 1$): Setting $r_{\mathbf{x}}[l] = 0$, Equation 4.47 implies that $\Delta_{\mathbf{x}}[l] = 1$. Thus, $\mathbf{x}$ lies inside the polyhedron l. On the other hand, setting $r_{\mathbf{x}}[l] < 0$, Equation 4.47 implies that $\Delta_{\mathbf{x}}[l] = 0$. Thus, $\mathbf{x}$ lies outside the polyhedron l.

For an unused polyhedron l ($\theta_l = 0$): Equation 4.52 implies that $\forall i \in \{1,\ldots,w\} : z_{l*w+i} = 0$. Thus, Equations 4.50–4.51 imply that $\forall i \in \{1,\ldots,w\} : \mathbf{A}[l*w+i,.] = 0 \ \wedge \ \mathbf{b}[l*w+i] = 0$. Hence, according to Equation 4.45, we can see that $\forall \mathbf{x} \in \Upsilon, \forall i \in \{1,\ldots,w\} : \delta_{\mathbf{x}}[l*w+i] = 0$, and consequently Equation 4.46 would imply that

$\forall \mathbf{x} \in \Upsilon : r_{\mathbf{x}}[l] = 0$. In this case Equation 4.47 implies that $\forall \mathbf{x} \in \Upsilon : \Delta_{\mathbf{x}}[l] = 0$. Thus, an unused polyhedron does not affect Equations 4.48 and 4.49.

Finally, the constraints of Equation 4.48 enforce the requirement that each state vector in the set $\widehat{P(\hat{S}_{rs})}$ should lie inside at least one of the constructed polyhedra, whereas the constraints of Equations 4.49 enforce the requirement that each of the state vectors in the set $\widehat{P(\hat{S}^b_{r\bar{s}})}$ should lie outside the constructed polyhedra.

Next we discuss the proper pricing of the parameters M_1 and M_2. It can be easily seen that M_1 can be set to $1+\epsilon$. On the other hand, to price M_2, we notice that the boundedness of the state vectors allows us to define $C = \sup_{\forall \mathbf{x} \in \Upsilon, j \in \{1,\ldots,n\}} \mathbf{x}[j]$. Moreover, taking into account the restriction of the elements of $\mathbf{A}$ and $\mathbf{b}$ to the interval $[0,1]$, the careful study of Equation 4.45 enables us to set M_2 equal to $n \cdot C$ during the solution of the considered formulation.

Theorem 4.4 *The application of the formulation of Equations 4.44–4.53 to the sets* $\widehat{P(\hat{S}_{rs})}$ *and* $\widehat{P(\hat{S}^b_{r\bar{s}})}$ *returns a set of packing polyhedra* Λ *that enables the establishment of a minimal Simple Boolean classifier for the two sets.*

An iterative algorithm for synthesizing a Simple Boolean classifier The main idea that underlies the proposed algorithm is to construct the sought classifier BC^Q_S by adding one packing polyhedron at a time; in particular, at each iteration we consider the set of points in $\widehat{P(\hat{S}_{rs})}$ that has not been separated yet from the set $\widehat{P(\hat{S}^b_{r\bar{s}})}$ by any of the constructed polyhedra, and we try to identify a polyhedron that will separate the maximum possible number of these points from $\widehat{P(\hat{S}^b_{r\bar{s}})}$. Next we present a MIP formulation that can support the computation of the packing polyhedron sought at each of the iterations described above. In fact, the presented MIP formulation supports both the computation of a packing polyhedron to separate safe states and the computation of a covering polyhedron to separate unsafe states according to the settings of the parameter "t". The main inputs for this formulation

are: (i) a set of projected safe state vectors, $S \subseteq \widehat{P(\hat{S}_{rs})}$, and (ii) a set of projected unsafe state vectors, $U \subseteq \widehat{P(\hat{S}^b_{r\bar{s}})}$. Some additional inputs that provide additional control are: (i) parameters w, ϵ, M_1 and M_2 that play a role similar to that played by the corresponding parameters in the MIP formulation of Equations 4.44-4.53, (ii) a parameter t that should be set to zero to compute a packing polyhedron, and to one to compute a covering polyhedron, and (iii) a parameter $\upsilon \in [0,1]$ that represents the relative weights of the components of the dual objective of the formulation; $\upsilon = 0$ if the objective is to minimize the number of linear inequalities of the synthesized polyhedron, whereas $\upsilon = 1$ if the objective is to maximize the number of the separated states. In the subsequent discussion, let $\Upsilon \equiv S \cup U$, and $n \equiv |L_P|$. The variables z_i, $(\mathbf{A}[i,.], \mathbf{b}[i])$, $\delta_{\mathbf{x}}[i]$, $\Delta_{\mathbf{x}}$, $r_{\mathbf{x}}$, and $\mathbf{x}$ employed by the proposed formulation play a role similar to that played by the corresponding variables in the MIP formulation of Equations 4.44-4.53. The formulation itself is expressed by the following equations:

$$\text{Min } (1-\upsilon) \cdot \sum_{i=1}^{w} z_i - \upsilon \cdot ((1-t) \cdot \sum_{\mathbf{s} \in S} \Delta_{\mathbf{s}} + t \cdot \sum_{\mathbf{u} \in U} \Delta_{\mathbf{u}}) \tag{4.54}$$

$$\forall \mathbf{x} \in \{\Upsilon\}, \forall i \in \{1, \ldots, w\} : \ (1-t) \cdot \epsilon + M_1 \cdot (\delta_{\mathbf{x}}[i] - 1) \leq \mathbf{A}[i,.] \cdot \mathbf{x} - \mathbf{b}[i] \leq M_2 \cdot \delta_{\mathbf{x}}[i] - t \cdot \epsilon \tag{4.55}$$

$$\forall \mathbf{x} \in \{\Upsilon\} : (1-2t) \cdot \sum_{i=1}^{w} \delta_{\mathbf{x}}[i] + w \cdot t + r_{\mathbf{x}} = 0 \tag{4.56}$$

$$\forall \mathbf{x} \in \{\Upsilon\} : w \cdot (\Delta_{\mathbf{x}} - 1) \leq r_{\mathbf{x}} \leq (\Delta_{\mathbf{x}} - 1) \tag{4.57}$$

$$\forall \mathbf{u} \in U : \Delta_{\mathbf{u}} \leq t \tag{4.58}$$

$$\forall \mathbf{s} \in S : \Delta_{\mathbf{s}} \leq (1-t) \tag{4.59}$$

Algorithm 4.3 The iterative construction of a Simple Boolean classifier

Input:(i) $\widehat{P(\hat{S}_{rs})}$; (ii) $\widehat{P(\hat{S}^b_{r\bar{s}})}$; (iii) υ; (iv) w; (v) ϵ, M_1, and M_2.
Output:A list $\Lambda = \{\Lambda_1, \ldots, \Lambda_\alpha\}$.

1: $S \leftarrow \widehat{P(\hat{S}_{rs})}$; $\quad i := 0$;
2: **while** $S \neq \emptyset$ **do**
3: $\quad i := i + 1$;
4: $\quad$ Generate a polyhedron $\Lambda_i = \langle \mathbf{A}_i, \mathbf{b}_i \rangle$ by solving the MIP formulation of Equations 4.54–4.62 with input S and $\widehat{P(\hat{S}^b_{r\bar{s}})}$ setting the parameter t to zero.
5: $\quad$ Remove the set $\{\mathbf{x} \in S \mid \mathbf{A}_i\mathbf{x} - \mathbf{b}_i \leq \mathbf{0}\}$ from S.
6: **end while**
7: $\alpha = i$; return $\langle \Lambda_1, \ldots, \Lambda_\alpha \rangle$;

$$\forall i \in \{1, \ldots, w\}, \forall j \in \{1, \ldots, n\} : 0 \leq \mathbf{A}[i, j] \leq z_i \tag{4.60}$$

$$\forall i \in \{1, \ldots, w\} : 0 \leq \mathbf{b}[i] \leq z_i \tag{4.61}$$

$$z_i,\ \delta_{\mathbf{x}}[i],\ \Delta_{\mathbf{x}}[l] \in \{0, 1\} \tag{4.62}$$

The mechanics of Equations 4.54–4.62 are very similar to those of Equations 4.44–4.53 with $k = 1$.

The complete algorithm that utilizes the formulation of Equations 4.54–4.62 for the iterative construction of a Simple Boolean classifier for the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$, is provided in Algorithm 4.3.

Theorem 4.5 *The application of Algorithm 4.3 to the sets* $\widehat{P(\hat{S}_{rs})}$ *and* $\widehat{P(\hat{S}^b_{r\bar{s}})}$ *corresponding to a given RAS* Φ*, returns a set of packing polyhedra* Λ *that enables the establishment of a Simple Boolean classifier for the original sets* S_{rs} *and* $S^b_{r\bar{s}}$.

Proof: Since the algorithm terminates when all the safe states are separated, and since a safe state is separated *iff* it is located inside at least one of the synthesized packing polyhedra, $\forall \mathbf{x} \in \widehat{P(\hat{S}_{rs})} : \exists i \in \{1, \ldots, \alpha\}$ s.t. $\chi_i = 1$. Therefore, $\forall \mathbf{x} \in$

$\widehat{P(\hat{S}_{rs})} : F(\chi(\mathbf{x},\Lambda_1),\ldots,\chi(\mathbf{x},\Lambda_\alpha)) = \bigvee_{i=1}^{\alpha}\chi_i = 1$. On the other hand, the constraints of Equation 4.58 imply that each unsafe state should lie outside any constructed polyhedron. Therefore, $\forall \mathbf{x} \in \widehat{P(\hat{S}^b_{r\bar{s}})}, \forall i \in \{1,\ldots,\alpha\} : \chi_i = 0$. Therefore, $\forall \mathbf{x} \in \widehat{P(\hat{S}^b_{r\bar{s}})}$: $F(\chi(\mathbf{x},\Lambda_1),\ldots,\chi(\mathbf{x},\Lambda_\alpha)) = \bigvee_{i=1}^{\alpha}\chi_i = 0$. Furthermore, applying Proposition 4.8 and 4.7, an effective separator for the sets S_{rs} and $S^b_{r\bar{s}}$ can be established. □

4.3.3.2 Alternating Boolean classifier

The idea of the Alternating Boolean classifier was motivated by Algorithm 4.3, where the set of safe states shrinks at each iteration, whereas the set of unsafe states remains constant throughout the algorithm. A computationally more efficient approach is to decrease the cardinalities of both sets, and the Alternating Boolean classifier is designed to support such an approach.

Definition 4.10 *An "*Alternating Boolean classifier*", BC^Q_A, is a Boolean classifier such that $\Lambda = \{\Lambda_1,\ldots,\Lambda_\alpha\}$, $\Psi = \{\Psi_1,\ldots,\Psi_{\alpha-1}\}$, and $F = \bigvee_{i=1}^{\alpha}(\chi_i(\bigwedge_{j=1}^{i-1}\kappa_j))$. Thus, $|BC^Q_A| = (2|L_P|+1)\cdot(\sum_{i=1}^{\alpha} m_{\Lambda_i} + \sum_{i=1}^{\alpha-1} m_{\Psi_i}) + 3\alpha$.*

Proposition 4.10 *Given the two sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$, there exists an Alternating Boolean classifier that separates the two subspaces.*

Proof: It is sufficient to show that the alternating classifier subsumes the simple classifier. Setting $\Psi_1 = \ldots = \Psi_{\alpha-1} = \{\mathbf{x} : \mathbf{0}^T\mathbf{x} \geq 1\}$, all the covering polyhedra are empty; then the alternating classifier reduces to a simple classifier. □

Algorithm 4.4 presents the complete procedure for synthesizing an Alternating Boolean classifier. To fix the structure of the supporting alternating classifier, the algorithm terminates when the set S empties. If the set U empties first, an additional iteration is performed to synthesize an additional packing polyhedron that empties the set S.

Proposition 4.11 *The application of Algorithm 4.4 to the sets $\widehat{P(\hat{S}_{rs})}$ and $\widehat{P(\hat{S}^b_{r\bar{s}})}$*

Algorithm 4.4 The iterative construction of an Alternating Boolean classifier

Input:(i) $\widehat{P(\hat{S}_{rs})}$; (ii) $\widehat{P(\hat{S}^b_{r\bar{s}})}$; (iii) υ; (iv) w; (v) ϵ, M_1, and M_2.
Output:A list $(\Lambda_1, \ldots, \Lambda_\alpha, \Psi_1, \ldots, \Psi_{\alpha-1})$;

1: $S \leftarrow \widehat{P(\hat{S}_{rs})}$; $U \leftarrow \widehat{P(\hat{S}^b_{r\bar{s}})}$; $i := 0$
2: **while** TRUE **do**
3: $i := i+1$;
4: Generate a packing polyhedron $\Lambda_i = \langle \mathbf{A}_i, \mathbf{b}_i \rangle$ by solving the MIP Formulation given by Equation 4.54–4.62 with input S and U setting the parameter t to zero.
5: Remove the set $\{\mathbf{s} \in S \mid \mathbf{A}_i\mathbf{s} - \mathbf{b}_i \leq 0\}$ from S.
6: If $(S = \emptyset)$ $\alpha = i$; return $(\Lambda_1 \ldots \Lambda_\alpha, \Psi_1, \ldots, \Psi_{\alpha-1})$;
7: Generate a covering polyhedron $\Psi_i = \langle \mathbf{A}_i, \mathbf{b}_i \rangle$ by solving MIP Formulation given by Equation 4.54–4.62 with input S and U setting the parameter t to one.
8: Remove the set $\{\mathbf{u} \in U \mid \mathbf{A}_i\mathbf{u} - \mathbf{b}_i \geq 0\}$ from U.
9: **end while**

returns the sets of polyhedra Λ and Ψ that enable the establishment of an Alternating Boolean classifier for the two sets.

Proof: Let S_i be the set of safe states separated by Λ_i, U_i be the set of unsafe states separated by Ψ_i, and U_α be the set of unsafe states not separated by any polyhedron. According to the mechanics of Algorithm 4.4, we can see that:

$$\forall \mathbf{x} \in S_i : \quad \chi_i = 1 \quad \wedge \quad \bigwedge_{j=1}^{i-1} \kappa_j = 1 \tag{4.63}$$

$$\forall \mathbf{x} \in U_i : \kappa_i = 0 \quad \wedge \quad \forall \mathbf{x} \in U_i,\ \forall k \in \{1, \ldots, i\} : \chi_k = 0 \tag{4.64}$$

$$\forall \mathbf{x} \in U_\alpha,\ \forall k \in \{1, \ldots, \alpha\} : \chi_k = 0 \tag{4.65}$$

Since $\bigcup_{i=1}^{\alpha} S_i = \widehat{P(\hat{S}_{rs})}$ and $\bigcup_{i=1}^{\alpha} U_i = \widehat{P(\hat{S}^b_{r\bar{s}})}$, it can be concluded that $\forall \mathbf{x} \in \widehat{P(\hat{S}_{rs})} : \bigvee_{i=1}^{\alpha}(\chi_i(\bigwedge_{j=1}^{i-1} \kappa_j)) = 1$, and that $\forall \mathbf{x} \in \widehat{P(\hat{S}^b_{r\bar{s}})} : \bigvee_{i=1}^{\alpha}(\chi_i(\bigwedge_{j=1}^{i-1} \kappa_j)) = 0$. □

4.3.4 Computational results

In this subsection, we report a set of experiments that demonstrates the applicability of the proposed methodologies for the construction of Boolean classifiers. First, we apply the methods introduced in this section to the RAS configuration defined in

Table 4.6: The results of the application of the Boolean classifier methods on the RAS configuration depicted in Table 4.3

$\widehat{P(\hat{S}_{rs})} = \{[3\ 3\ 0\ 0\ 0\ 3]^T,\ [1\ 3\ 0\ 0\ 1\ 3]^T,\ [0\ 3\ 1\ 2\ 0\ 0]^T,$ $[0\ 3\ 0\ 3\ 0\ 3]^T,\ [0\ 3\ 0\ 2\ 1\ 3]^T,\ [0\ 2\ 1\ 2\ 0\ 3]^T\}$ $\widehat{P(\hat{S}^b_{r\bar{s}})} = \{[0\ 3\ 1\ 0\ 0\ 1]^T,\ [0\ 0\ 0\ 3\ 1\ 0]^T,\ [1\ 0\ 0\ 1\ 0\ 0]^T\}$
The Simple Boolean classifier obtained by the formulation of Equations 4.44–4.53
$\Lambda_1 = \{\mathbf{x} : x_2 + x_3 \leq 0\}$, $\Lambda_2 = \{\mathbf{x} : x_0 + 0.033 \cdot x_1 + x_2 + 0.3 \cdot x_3 + 0.2 \cdot x_4 \leq 1\}$ $\Lambda_3 = \{\mathbf{x} : x_0 + 0.3 \cdot x_1 + x_4 + 0.1 \cdot x_5 \leq 0.9\}$ $F = \chi_1 \vee \chi_2 \vee \chi_3$
The Simple Boolean classifier obtained by Algorithm 4.3
$\Lambda_1 = \{\mathbf{x} : 0.75 \cdot x_0 + 0.0167 \cdot x_1 + x_2 + 0.3 \cdot x_3 + 0.15 \cdot x_4$ $\leq 0.95\}$, $\Lambda_2 = \{\mathbf{x} : x_0 + 0.3 \cdot x_1 + x_4 + 0.1 \cdot x_5 \leq 0.9\}$, $\Lambda_3 = \{\mathbf{x} : 0.1 \cdot x_2 + 0.1 \cdot x_3 \leq 0\}$, $F = \chi_1 \vee \chi_2 \vee \chi_3$
The Alternating Boolean classifier obtained by Algorithm 4.4
$\Lambda_1 = \{\mathbf{x} : 0.75 \cdot x_0 + 0.0167 \cdot x_1 + x_2 + 0.3 \cdot x_3 + 0.15 \cdot x_4$ $\leq 0.95\}$, $\Lambda_2 = \{\mathbf{x} : 0 \cdot x_1 \leq 0\}$, $\Psi_1 = \{\mathbf{x} : 0.1 \cdot x_2 + x_3 \geq 0.1;\ x_0 + 0.3 \cdot x_1 + x_4 + 0.1 \cdot x_5 \geq 1\}$, $F = \chi_1 \vee (\chi_2 \wedge \kappa_1)$

Table 4.3. As mentioned in Subsection 4.2.5, applying the introduced thinning techniques on the underlying reachable state space led to a set $\widehat{P(\hat{S}_{rs})}$ with 6 states, and a set $\widehat{P(\hat{S}^b_{r\bar{s}})}$ with 3 states, whereas the dimensionality of the projected subspace V_P is 6. These sets are reported again in Table 4.6. Furthermore, Table 4.6 reports the Boolean classifiers obtained by applying (i) Equations 4.44–4.53, (ii) Algorithm 4.3, and (iii) Algorithm 4.4. The Simple Boolean classifiers obtained by the first two methods have the same size which equals to $(2|L_P|+1) \cdot (\sum_{i=1}^{\alpha} m_{\Lambda_i}) + \alpha = (2 \cdot 6 + 1) \cdot 3 + 3 = 42$. On the other hand, the size of the Alternating Boolean classifier obtained by the third method equals to $(2|L_P|+1) \cdot (\sum_{i=1}^{\alpha} m_{\Lambda_i} + \sum_{i=1}^{\alpha-1} m_{\Psi_i}) + 3\alpha = (2 \cdot 6 + 1) \cdot 3 + 3 \cdot 2 = 43$.

Table 4.7: A sample of our experimental results for the construction of Boolean classifiers

ξ	$\lvert S_{rs}\rvert$	$\lvert S_{r\bar{s}}\rvert$	$\lvert\widehat{P(\hat{S}_{rs})}\rvert$	$\lvert\widehat{P(\hat{S}^b_{r\bar{s}})}\rvert$	$\lvert L_P\rvert$	$\lvert BC^Q_S\rvert_E$	$\lvert BC^Q_S\rvert_H$	$\lvert BC^A_S\rvert$
24	109	1379	34	35	17	107	215	182
24	593	2756	31	32	14	176	297	274
36	623	1133	69	40	26	161	269	163
16	342	857	30	23	11	117	143	171
32	758	2611	71	31	19	158	239	202
35	222	1244	56	68	27	167	391	395
26	196021	11451	46	15	8	121	178	129
24	94431	72238	849	71	17	-	4929	1166
24	115766	28510	95	12	11	-	402	220
48	10913	96361	404	163	41	-	5770	1174
48	6596	129036	527	221	39	-	7152	2012
24	104550	49620	1937	67	15	-	255	292
40	42571	69016	3754	188	30	-	7098	1687
39	757699	700781	2120	89	21	-	7329	1468
56	21099	906478	741	225	48	-	-	2659
63	65992	2105590	2314	708	56	-	-	9503
64	34695	1773193	1697	427	55	-	-	5287
64	118470	1253425	3432	612	54	-	-	8508
88	53080	1410311	5560	1046	74	-	-	11737
52	1622861	2600349	15783	257	37	-	-	3442
65	396931	1146292	13742	559	51	-	-	8882

Table 4.7 reports the results obtained from the application of the methods proposed in this section for the deployment of the maximally permissive DAP on additional 21 RAS configurations. More specifically, for each of these configurations, Table 4.7 reports: (i) the total number of processing stages ξ, (ii) the cardinality of the reachable safe subspace S_{rs}, (iii) the cardinality of the reachable unsafe subspace $S_{r\bar{s}}$, (iv) the cardinality of the set of maximal projected reachable safe states $\widehat{P(\hat{S}_{rs})}$, (v) the cardinality of the set of minimal projected boundary reachable unsafe states $\widehat{P(\hat{S}^b_{r\bar{s}})}$, (vi) the dimensionality of the projected subspace V_P, (vii) the size of the Simple Boolean classifier obtained by applying the formulation of Equations 4.44–4.53, (viii) the size of the Simple Boolean classifier obtained by applying Algorithm 4.3, and (ix) the size of the Alternating Boolean classifier obtained by applying Algorithm 4.4.

The "-" entry indicates that the relevant methodology failed to find a solution within a 120 minutes budget.[4] The following observations can be drawn from the

[4]Relaxing the time constraint did not help in finding a solution. For the sake of completeness, we also report that these experiments were performed on a 2.66 GHz quad-core Intel Xeon 5430

reported results: (i) As expected, the size of the classifier obtained using the formulation of Equations 4.44–4.53 is smaller than that obtained using the other two methods. (ii) On the other hand, Algorithm 4.3 is capable of solving large problem instances that can not be practically solved using the formulation of Equations 4.44–4.53. (iii) Finally, Algorithm 4.4 is capable of solving large problem instances that can not be practically solved using either Algorithm 4.3 or the formulation of Equations 4.44–4.53.

4.4 Concluding remarks

This chapter has extended the results of Chapter 3, that sought the representation of the maximally permissive DAP for the RAS class of Definition 1.1 as a parsimonious classifier effecting the dichotomy of the underlying reachable state space into its safe and unsafe subspaces, to the case where the sought dichotomy cannot be represented by a linear classifier. We have proposed new classification schemes for this more complex case and established formally their completeness, i.e., their ability to provide an effective classifier for every instance of the considered RAS class. We have also provided computationally efficient procedures for the synthesis of the sought classifiers. Finally, the effectiveness and the efficacy of the presented approaches have been demonstrated and assessed through a series of computational experiments.

Closing the discussion on our computational experiments, we want also to notice that none of the RAS configurations employed in the experiments presented in this chapter accepts a characterization of its maximally permissive DAP as a set of linear inequalities, and therefore, they are not amenable to the Petri net-based methodologies discussed in Chapter 1. This effect was verified through techniques similar to those reported in Chapter 3, and it manifests another powerful attribute of the approaches considered in this chapter with respect to other approaches reported in

processor with 6 MB of cache memory and 32 GB RAM; however, each job ran on a single core.

the current literature.

CHAPTER V

NON-PARAMETRIC CLASSIFIERS

5.1 Introduction

Similar to Chapter 4, we address the classifier design problem in the context of the RAS class defined in Chapter 1. However, in this chapter, we adopt a non-parametric representation for the sought classifier. As pointed out in Chapter 1, the effective implementation of the maximally permissive DAP for any given RAS, Φ, is equivalent to the recognition and the blockage of transitions from the safe to the unsafe region of the underlying state space S. Thus, in principle, an implementation of the maximal DAP for any given RAS Φ can be based on

- the explicit enumeration and storage of the set $S^b_{r\bar{s}}$, and
- a single-step lookahead scheme that, starting from the initial state $\mathbf{s}_0$, enables any transition $\mathbf{s}' = f(\mathbf{s}, e)$ that is system-enabled according to Equation 1.1, only if $\mathbf{s}' \notin S^b_{r\bar{s}}$.

Motivated by the above remark, once the FSA that models the given RAS behavior has been obtained, the proposed non-parametric classification scheme seeks to encode the information necessary to resolve the underlying state safety problem in a "data structure / mechanism" sufficiently compact so that the problem can be effectively addressed within the time and the other resource constraints that typically arise in a real-time computation.

In the light of the above, the rest of the chapter is organized as follows: Section 5.2 presents the overall methodology pursued in this chapter for the construction of a non-parametric classifier. Section 5.3 presents the algorithm utilized for the construction

of the particular data structure that encodes the state space dichotomy, whereas Section 5.4 presents the algorithm utilized for the online assessment of the state safety property using the synthesized non-parametric classifier. Section 5.5 demonstrates the applicability of the approach by implementing it on an example problem instance, and it also reports our experiences with implementations involving larger and/or more complex RAS configurations. Finally, Section 5.6 concludes the chapter.

5.2 The non-parametric classifier design problem

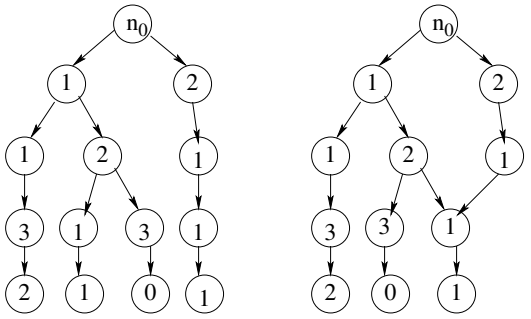
As pointed out in the previous section, the control scheme of the proposed non-parametric classifier requires the storage of of the set $S^b_{r\bar{s}}$ in such a manner that the test $\mathbf{s}' \notin S^b_{r\bar{s}}$ is tractable within the time budget constraints that are enforced by the "embedded / real-time" nature of the implemented supervisor. The rest of this section discusses how to facilitate this requirement and render the above control scheme a viable solution for many practical application contexts.

Obtaining a more compressed characterization of the set $S^b_{r\bar{s}}$ The data set $S^b_{r\bar{s}}$ can be further compressed using the thinning techniques introduced in Section 2.5. More specifically, using the monotonicity property of Proposition 2.1, we can explicitly store only the subset of the minimal elements, $\widehat{S}^b_{r\bar{s}}$, and during the online stage, given a state $\mathbf{s}$, we should check whether there exists a state $\mathbf{u}' \in \widehat{S}^b_{r\bar{s}}$ such that $\mathbf{s} \succeq \mathbf{u}'$. In other words, $\mathbf{s}$ is a safe state *iff* $\forall \mathbf{u}' \in \widehat{S}^b_{r\bar{s}},\ \exists i_{u'} \in L$ s.t. $\mathbf{s}[i_{u'}] < \mathbf{u}'[i_{u'}]$. From the definition of the set of dimensions L_0 in Section 2.5, we can see that $i_{u'} \notin L_0$; hence, the set of dimensions L_0 can be dropped from the stored data set, and therefore, the set $\widehat{P(\widehat{S}^b_{r\bar{s}})}$ obtained by applying the workflow depicted in Figure 2.2, contains sufficient information for the classification of a given state vector. Hence, given a vector $\mathbf{x}$, $\mathbf{x}$ is classified as a safe state *iff* it satisfies the following equation:

$$\forall \mathbf{u}' \in \widehat{S}^b_{r\bar{s}},\ \exists i_{u'} \in L_P \text{ s.t. } \mathbf{x}[i_{u'}] < \mathbf{u}'[i_{u'}] \tag{5.1}$$

The thinning procedure described above involves the post-processing of the set $S^b_{r\bar{s}}$ through some very simple and efficient computation. On the other hand, it leads to an extensive (frequently dramatic) reduction of the information that must be explicitly stored and processed for the effective implementation of the proposed control scheme. In fact, for many practical cases, a simple array-based storage of the elements of $\widehat{P(\widehat{S}^b_{r\bar{s}})}$ will be quite adequate for effecting the on-line computation that is involved in the implementation of the maximal DAP described in the previous paragraphs. However, in the rest of this section, we also discuss an additional data structure that can lead (i) to more efficient storage of the set $\widehat{P(\widehat{S}^b_{r\bar{s}})}$ and (ii) to more expedient algorithms for the on-line test suggested by Equation 5.1. The relevant gains are further demonstrated and assessed through the numerical experimentation reported in Section 5.5.

Storing the set $\widehat{P(\widehat{S}^b_{r\bar{s}})}$ through n-ary decision diagrams The (n-ary) decision diagrams proposed in the context of this chapter for the storage and on-line processing of the set $\widehat{P(\widehat{S}^b_{r\bar{s}})}$, is an adaptation of the concept of the binary decision diagram (BDD) that has been used for the efficient storage and manipulation of Boolean functions [13]. They can be systematically introduced by first defining the (n-ary) *decision tree* for the storage of k l-dimensional vectors $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$: This tree has a dummy root node, n_0, of depth 0, and l layers of nodes with depths from 1 to l that correspond to the l dimensions of vectors $\mathbf{v}_i$. Starting with node n_0 as the single node of layer 0, the tree nodes at each of the remaining layers are defined recursively as follows: The children of a node n at layer $l(n) \in \{0, \ldots, l-1\}$ correspond to all the possible values of coordinate $l(n)+1$ in the vector subset of $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ that is obtained by fixing the first $l(n)$ coordinates at the values specified by the path from the root node n_0 to node n. The coordinate value that corresponds to each node n in layers 1 to l, according to this node generation scheme, is characterized as the "content" of n. Obviously, the nodes generated for layer l according to the previous



(a) The decision tree (b) The corresponding decision diagran

Figure 5.1: A decision tree and the corresponding decision diagram storing the vector set $\{[1,2,1,1]^T, [2,1,1,1]^T, [1,1,3,2]^T, [1,2,3,0]^T\}$.

recursion have no children, and they constitute the leaf nodes of the tree. In the resulting tree structure, every vector $\mathbf{v}_i$, $i = 1, \ldots, k$, is represented by the path to one of the leaf nodes. We should also notice that the n-ary decision tree introduced in this paragraph, is a standard tool for efficient string storage and retrieval; typically, it is characterized as the *"TRIE"* (data) structure, and there are many variations of it and a considerable literature investigating their properties (cf. [7] and the references cited therein).[1]

The decision tree described in the previous paragraph is converted to a decision diagram by iteratively identifying and eliminating duplicate sub-graphs in the generated structure, while starting from the last layer l. Two subgraphs – or sub-diagrams – originating at given layer $i \in \{1, \ldots, l\}$ are considered duplicate if (i) they are isomorphic

[1] "TRIE" is supposed to stand for "reTRIEval". We also note that the literature contains some additional attempts to use the BDD concept and its extensions/ variations not only for data storage, but for the representation and analysis of DES-related dynamics; cf., for instance, the works of [18, 94]. In this work, we place the emphasis primarily on the storage and the retrieval efficiencies that are supported by this type of data structures. The computation of the stored content itself is based on techniques that are motivated by and customized to particular attributes of the considered application, and as discussed in the previous parts of this manuscript, they lead to additional informational compression and storage economies.

Algorithm 5.1 The algorithm constructing the n-ary decision diagram that provides a parsimonious representation for a given set of vectors V.

Input:A set of l-dimensional vectors $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$.
Output: An n-ary decision diagram providing a parsimonious representation for V.

1: $(n_0,\ n_f)$:= Initial Decision Diagram (V);
2: Downwards Compression (n_0);
3: Upwards Compression (n_f);

Procedure 5.2 Initial Decision Diagram (V)

Input:A set of l-dimensional vectors $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$.
Output:A node n_0 and a node n_f.

1: Construct the two dummy nodes n_0 and n_f;
2: **for** $i := 1 : k$ **do**
3: p:=new Node; $p.val := \mathbf{v}_i[1]$;
4: Add$(p, n_0.children)$; Add$(n_0, p.parents)$;
5: **for** $j := 2 : l$ **do**
6: q:=new Node; $q.val := \mathbf{v}_i[j]$;
7: $p.children := q$; Add$(p, q.parents)$;
8: $p := q$;
9: **end for**
10: $p.children := n_f$; Add$(p, n_f.parents)$;
11: **end for**
12: **return** $(n_0,\ n_f)$;

and (ii) each isomorphically related pair of nodes has the same content. Figure 5.1 exemplifies the above definitions by depicting the decision tree and the corresponding decision diagram that store the vector set $\{[1,2,1,1]^T, [2,1,1,1]^T, [1,1,3,2]^T, [1,2,3,0]^T\}$.

5.3 The algorithmic construction of n-ary decision diagrams

Algorithm 5.1 presents a systematic construction of an n-ary decision diagram for the parsimonious representation of a set of k l-dimensional vectors $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$. The data structure employed by this algorithm in order to represent a node of the derived decision diagram consists of (i) an integer field for storing the node "content", and (ii) two pointer lists that respectively provide the parents and the children of this

Procedure 5.3 Downwards Compression (n_0)

Input:A node n_0.
Output:A node n_0.

```
 1: Q := NIL; { an empty queue of nodes }
 2: Enqueue(n_0,Q);
 3: while Q! = NIL do
 4:    p := Dequeue(Q);
 5:    q := p.children;
 6:    while q! = NIL do
 7:       AddtoQ := FALSE;
 8:       r := q.next;
 9:       while r! = NIL do
10:          if q.val == r.val then
11:             AddChildren(r, q);
12:             Remove1(r, p.children);
13:             AddtoQ :=TRUE;
14:          end if
15:          r++
16:       end while
17:       if AddtoQ then
18:          Enqueue(q,Q);
19:       end if
20:       q++;
21:    end while
22: end while
```

node in the constructed diagram.[2] The overall computation of the considered algorithm is organized in three major phases: The first phase, depicted in Procedure 5.2, constructs an acyclic digraph with a single source node n_0 and a single sink node n_f, and with its internal nodes encoding the vectors $\mathbf{v}_i \in V$ as distinct, non-overlapping paths from n_0 to n_f. Clearly, the digraph produced at this phase is layered, with l internal layers. Furthermore, each of these internal layers contains $k(\equiv |V|)$ distinct nodes, and each such node has a single parent and a single child in the digraph. Moreover, each such internal node is labeled by the numerical value of the component that it represents in the considered vector set V; as already mentioned, this label defines

[2]Since a decision diagram is an acyclic graph, both of these two concepts make sense for each node. On the other hand, since the derived structure is a diagram and not a tree, each node can have more than one parent in it.

Procedure 5.4 Upwards Compression (n_f)

Input:A node n_f.
Output:A node n_f.

```
 1: Q := NIL; { an empty queue of nodes }
 2: Enqueue(n_f,Q);
 3: while Q! = NIL do
 4:    p := Dequeue(Q);
 5:    q := p.parents;
 6:    while q! = NIL do
 7:       AddtoQ := FALSE;
 8:       r := q.next;
 9:       while r! = NIL do
10:          if Equivalent(q, r) then
11:             AddParents(r, q);
12:             Remove2(r);
13:             AddtoQ :=TRUE;
14:          end if
15:          r++;
16:       end while
17:       if AddtoQ then
18:          Enqueue(q,Q);
19:       end if
20:       q++;
21:    end while
22: end while
```

the "content" of the node. The second phase of Algorithm 5.1, depicted in Procedure 5.3, is an operation of "downwards compression". This part of the computation essentially seeks to identify all the nodes in the diagram generated by Phase I that correspond to identical prefixes for the vectors passing through them, and merge them in a single node. Finally, the third phase of Algorithm 5.1, depicted in Procedure 5.4, is an operation of "upwards compression". This part of the computation seeks to identify nodes in the diagram generated by Phase II that correspond to the same sets of possible suffixes for the state vectors passing through them, and merge them.

In more technical terms, the downwards compression, that is performed in Procedure 5.3, is attained as follows: The digraph constructed in Procedure 5.2 is traversed on a layer-by-layer basis, starting from the source node n_0, and at each visited node,

all of its children with equal labels are merged to a single node. In particular, the function AddChildren(r, q) is meant to add the child of node r to the children of node q, and to redefine q as the parent of this added child.[3] Subsequently, the function Remove1$(r, p.children)$ removes node r from the children of node p, and releases the corresponding memory. This traversal continues until a layer is reached where no node merging occurs; at that point, Q will become empty, and the procedure will exit the loop of Line 3, which is the main loop for this procedure. The reader should notice that as a result of the performed nodal merging, some of the internal nodes will have more than one child at the end of Procedure 5.3, but this merging preserves the single-parent property for the internal nodes. Hence, it is clear that each cluster of merged nodes corresponds to the same vector prefix, as initially stated, and the performed merging does not distort the "information content" of the graph; i.e., the vector set that is defined by all the paths from the source to the sink node remains equal to V.

On the other hand, during its third phase, i.e., Procedure 5.4, the considered algorithm traverses again the digraph obtained in Procedure 5.3 on a layer-by-layer basis, but this time, the traversal starts from the sink node n_f. At each visited node, the procedure checks the list of its parents, to see whether there are equivalent nodes. Two nodes q and r are characterized as "equivalent" if they have the same content and the same lists of children; the relevant testing is performed by function Equivalent(q, r). A simple induction on the number of traversed layers can show that node equivalence essentially implies the same sets of possible suffixes for all the paths passing through them, and therefore, the nodal merging that is performed by the procedure is consistent with the phase objective that is stated at the opening paragraph of this section. The merging of the equivalent nodes q and r is performed

[3]The fact that node r has only one child is a consequence of the structure of the diagram that is returned by Procedure 5.2, and the graph traversal pattern that is applied in Procedure 5.3.

Table 5.1: The set of vectors used in the Example of Figure 5.2

$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_4$	$\mathbf{v}_5$
1	1	1	1	1
0	0	0	0	1
1	0	0	0	0
0	2	1	0	0
0	1	1	1	0
0	0	1	2	0
2	2	2	0	2

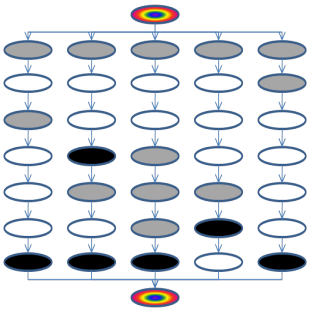
by the following two functions: Function AddParents(r, q) adds the parent of node r to the parents of node q, and also it updates the list of children of this added node by replacing node r in it by node q. Subsequently, the function Remove2(r) removes node r from the parent lists of all its children, and releases the memory allocated for the storage of this node.[4] Similarly to Procedure 5.3, the graph traversal described above terminates when a layer is encountered where no node merging takes place. Clearly, the information content of the digraph is also preserved under this new phase of node merging. Hence, Algorithm 5.1 is correct.

Figure 5.2 exemplifies Algorithm 5.1 by depicting the decision diagrams that result after the execution of each of its three phases on the vector set of Table 5.1. We also notice that while the dummy terminal node n_f is useful for the algorithmic construction of the decision diagram according to the logic described above, it does not play any substantial role during the on-line use of this diagram, and therefore, eventually it can be removed; this is the version of decision diagrams that we shall consider in the sequel.

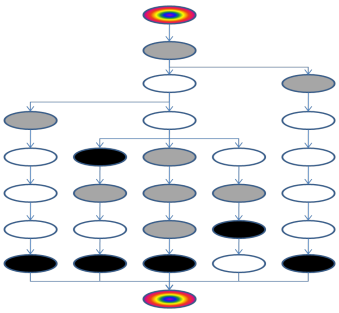
Complexity analysis: First, we notice that Procedure 5.2 runs in time $\mathcal{O}(kl)$, and results in a diagram with $\mathcal{O}(kl)$ nodes.

To analyze the computational complexity of Procedure 5.3, for every layer $j =$

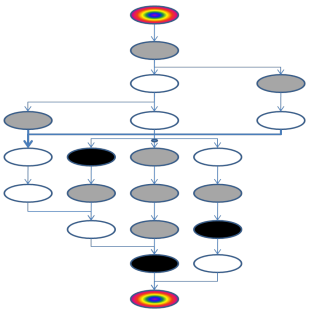
[4]Note that since, by the notion of equivalence, q and r have the same children, there is no need to add q in the parent lists of the children of r.



(a) Initial decision diagram



(b) Downwards compression



(c) Upwards compression

Figure 5.2: The decision diagrams produced at the end of each of the three major phases of Algorithm 5.1 when applied to the vector set of Table 5.1. The node content is encoded as follows: white corresponds to 0, grey to 1, and black to 2. The dummy nodes n_0 and n_f are depicted as multi-colored nodes.

$1, \ldots, l$, let us consider the equivalence relation $\mathcal{Q}_j$ that is defined on the vector set V by the equality of the vector prefixes across the layers 1 to $j-1$; i.e., the equivalence classes Θ_j^i of $\mathcal{Q}_j$ consist of all the vectors in V that have their first $j-1$ coordinates equal. For every such equivalence class Θ_j^i of the partitioning $\mathcal{Q}_j$ of V, let us also consider the set consisting of the distinct values of the j-th coordinate for the vectors in Θ_j^i, and let c_j^i denote the cardinality of that set. Finally, set $c \equiv \max_{j,i}\{c_j^i\}$. From a more intuitive standpoint, c defines the maximum possible number of siblings (i.e., children of the same parent) that can exist in the diagram constructed by Procedure 5.3. Then, some additional remarks that are important for characterizing the computational complexity of Procedure 5.3 are the following: During the processing

of any nodal layer $j \in \{1, \ldots, l\}$ by this procedure, the corresponding nodes are partitioned into sibling classes, and from the previous discussion, it follows that the node contents of every such class can take at most c distinct values. This last remark, when combined with the fact that every merged node is removed from the children list of its parent, further implies that the while loop of Line 6 will be executed at most c times for every class. Also, every such execution will go through the while loop of Line 9, a number of times that is no greater than the number of siblings in that class. Hence, the number of executions of the while loop of Line 9 across all the sibling classes of any single layer of the considered diagram will be $\mathcal{O}(kc)$; and when considered across all layers, this number will be $\mathcal{O}(lkc)$. To obtain the complete characterization of the computational complexity of Procedure 5.3, it remains to characterize the computational complexity of the operations involved in the merging of the nodes q and r, i.e., the computational complexity of the functions AddChildren(r,q) and Remove1$(r,p.children)$. Obviously, under the pointer-based implementation of the lists employed by the algorithm, the complexity of Remove1$(r,p.children)$ is $\mathcal{O}(1)$. The complexity of AddChildren(r,q) is also $\mathcal{O}(1)$, since as already remarked, node r will have only a single child, and this child does not belong to the current children of node q. Hence, the overall complexity of Procedure 5.3 remains $\mathcal{O}(lkc)$.

The computational complexity of Procedure 5.4 can be obtained through analysis similar to that performed for the complexity of Procedure 5.3. Some key observations that lead to the characterization of this complexity are as follows: During the execution of this procedure, for every nodal pair (r,q) appearing in the function AddParents(r,q), node r has a single parent that is different from the parent of node q, and this parent node of r has at most c children (since nodes q and r are equivalent, and the diagram obtained by Procedure 5.3 has a TRIE structure). Hence, the complexity of function AddParents(r,q) is $\mathcal{O}(c)$. Obviously, the complexity of function Remove2(r) is also $\mathcal{O}(c)$ (since the number of children of a node does not

increase in this procedure). Finally, the evaluation of the function Equivalence(q, r) is also $\mathcal{O}(c)$ (since equivalent nodes have the same nodal content, and the same children lists). Hence, the entire computational complexity of an iteration of the while loop of Line 9 is $\mathcal{O}(c)$. An analysis similar to that performed for the complexity of Procedure 5.3 can establish that the aforementioned loop will be executed $\mathcal{O}(lk^2)$ times by the procedure. Hence, the entire computational complexity of this procedure is $\mathcal{O}(lk^2c)$.

Since Procedure 5.4 has the highest computational complexity in Algorithm 5.1, its complexity defines also the computational complexity of the entire algorithm.

5.4 The on-line implementation of the maximally permissive DAP through n-ary decision diagrams

The employment of the n-ary decision diagram in the context of the single-step lookahead control scheme described at the beginning of this section, is supported through Algorithm 5.5. More specifically, Algorithm 5.5 takes as input the decision diagram of a vector set V and a vector $\mathbf{v}'$, and it checks whether there is a vector $\mathbf{v} \in V$ such that $\mathbf{v} \preceq \mathbf{v}'$. Starting with the root dummy node n_0, this algorithm essentially performs a depth-first search for a path to a leaf node, such that, at every layer $j = 1, \ldots, l$, it engages a node with content no greater than the value of component $\mathbf{v}'[j]$. If such a path is identified, the algorithm returns 'TRUE', (i.e., $\exists \mathbf{v} \in V$ such that $\mathbf{v} \preceq \mathbf{v}'$, namely, the vector defined by the node contents of the constructed path). In the opposite case, the algorithm returns 'FALSE'.

Complexity analysis: The worst case computational complexity of this algorithm is $\mathcal{O}(\bar{n})$, where $\bar{n}$ denotes the number of nodes in the decision diagram of V.

Algorithm 5.5 An algorithm that takes as input the decision diagram of a vector set V and a vector $\mathbf{v}'$, and checks whether there is a vector $\mathbf{v} \in V$ such that $\mathbf{v} \preceq \mathbf{v}'$.

Input: The decision diagram of a vector set V and a vector $\mathbf{v}'$.
Output:A Boolean variable indicating whether there is a vector $\mathbf{v} \in V$ such that $\mathbf{v} \preceq \mathbf{v}'$.

```
1: l̄ := dim(v);  EXIT := FALSE;
2: Push (n_0, 0) on SearchStack;
3: while SearchStack ≠ ∅ ∧ ¬EXIT do
4:    (n, l) ← pop SearchStack;
5:    l := l + 1;
6:    for each child n' of n do
7:       if content(n') ≤ v'[l] ∧ ¬EXIT then
8:          if l = l̄ then
9:             EXIT := TRUE
10:         else
11:            push (n', l) onto SearchStack;
12:         end if
13:      end if
14:   end for
15: end while
16: Return EXIT;
```

5.5 Computational results

In this section we report a number of computational experiments that demonstrate the efficacy of the proposed approach and assess its applicability in the context of the RAS class considered in this chapter. We begin the presentation of these results by considering the application of our methodology to the synthesis of the maximally permissive DAP for the RAS configuration defined in Table 5.2. The considered RAS has seven resource types, $\{R_1, \ldots, R_7\}$, each with capacity $C_i = 3$. It also has four process types, $\{J_1, J_2, J_3, J_4\}$, with each process type defined by the linear route provided in Table 5.2. In particular, each of the depicted routes constitutes a sequence of processing stages with each stage engaging a single resource type at the amount indicated by the corresponding coefficient; for instance, the first processing stage of the first process type engages a single unit of resource R_2, the second stage of the same type engages two units of resource R_7, etc. Hence, the depicted RAS has 20

Table 5.2: The RAS considered in the example of Section 5.5

Resource Types:	$\{R_1, R_2, \ldots, R_7\}$
Resource Capacities:	$C(R_i) = 3,\ \forall i \in \{1, 2, \ldots, 7\}$
Process Types:	$\{J_1, J_2, J_3, J_4\}$
Process Routes:	$\mathcal{G}_1 : R_2 \rightarrow 2R_7 \rightarrow 2R_4 \rightarrow 2R_3$ $\mathcal{G}_2 : 3R_4 \rightarrow 3R_2 \rightarrow 2R_4 \rightarrow 3R_7 \rightarrow$ $2R_3 \rightarrow R_5 \rightarrow R_4$ $\mathcal{G}_3 : R_1 \rightarrow 2R_5 \rightarrow 3R_1 \rightarrow R_7 \rightarrow 2R_6$ $\mathcal{G}_4 : 2R_2 \rightarrow R_3 \rightarrow R_4 \rightarrow 3R_2$

distinct processing stages, in total, and this number defines the dimensionality of its state vector according to the definitions provided in Chapter 1.

The application of the DAP design methodology proposed in this chapter revealed that the considered RAS has a reachable state space of 351,604 states, with 135,414 of them being safe states and the remaining 216,190 being unsafe states. Among the boundary reachable unsafe states, 49 of them are minimal. In these 49 minimal unsafe states, 6 out of the 20 state components are always equal to zero, and therefore, the dimensions that need to be explicitly stored are only the remaining 14. The corresponding decision diagram has 180 nodes and it is depicted in Figure 5.3. The storage of this diagram in a core computer memory requires 399 integer locations, 180 of these locations for storing the nodal content itself, and the remaining 219 locations for storing the pointers that correspond to the arcs of this diagram. On the other hand, the storage of the 49 14-dim vectors of $\widehat{P(\widehat{S}^b_{r\bar{s}})}$ in a 2-dim array involves $49 \times 14 = 686$ integer entries. Therefore, the alternative storage mechanism of Figure 5.3 utilizes only a little more than 58% of the storage space utilized by the array-based storage of $\widehat{P(\widehat{S}^b_{r\bar{s}})}$.

Table 5.3 reports the results that were obtained from the application of the proposed method for the deployment of the maximally permissive DAP on 10 additional

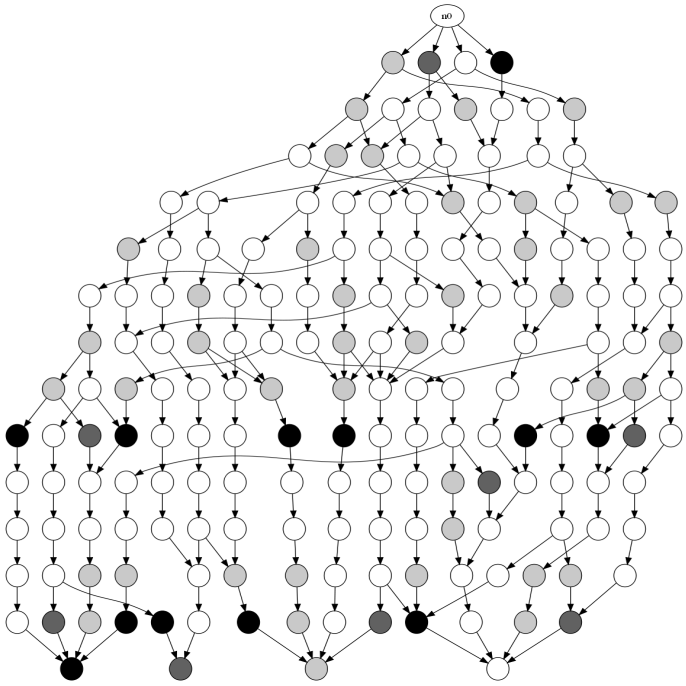


Figure 5.3: The acyclic digraph storing the vector set $\widehat{P(\hat{S}^b_{r\bar{s}})}$ for the example RAS of Table 5.2. The white, light gray, dark gray and black nodes correspond to nodes having respective "content" values of 0, 1, 2 and 3.

RAS configurations. More specifically, for each of these configurations, Table 5.3 reports: (i) the total number of processing stages (and therefore, the dimensionality of the corresponding state space); (ii) the cardinality of the reachable safe subspace S_{rs}; (iii) the cardinality of the reachable unsafe subspace $S_{r\bar{s}}$; (iv) the cardinality of the set of minimal boundary reachable unsafe states $\widehat{S}^b_{r\bar{s}}$; (v) the dimensionality of the projected subspace V_P; (vi) the number of integer entries that would be necessary for the storage of the elements of $\widehat{P(\widehat{S}^b_{r\bar{s}})}$ in a 2-dim array – the entries of this column are obtained by multiplying the entries of the previous two columns in the table; (vii) the number of the nodes employed by the corresponding decision diagram; (viii) the total storage capacity, in terms of integer entries, that is required for the storage of the entire structure of the corresponding decision diagram; (ix) the storage compression attained by the n-ary decision diagram – the entries of this column are obtained by taking the ratio of the entries in columns (viii) and (vi) of the depicted table, i.e.,

the storage compression is measured as the ratio of the storage requirements posed by the n-ary decision diagram to the storage requirements that would be necessary in case of a 2-dim array-based representation of the vector set $\widehat{P(\widehat{S}^b_{r\bar{s}})}$; (x) the total computational time, in seconds, that is required for the construction of the decision diagram for each of the listed cases. Figure 5.4 also depicts the attained storage compression for an even broader data set of 42 RAS configurations (this data set includes the configurations involved in Table 5.3).

The results of Table 5.3 and Figure 5.4 corroborate that the proposed method is effectively applicable to RAS with very large state spaces and it can lead to a very compact representation of the corresponding maximally permissive DAP. The informational compression and the corresponding storage gains that are attained by the application of the n-ary decision diagrams are substantial - in most of the reported cases the attained compression is more than 80%. Furthermore, the plot of Figure 5.4 reveals that the attained storage efficiencies become more prominent as the storage requirements for the more conventional, array-based representation of the set $\widehat{P(\widehat{S}^b_{r\bar{s}})}$ become larger.

Finally, we also notice that none of the RAS configurations employed in the presented experiments accepts a characterization of its maximally permissive DAP as a set of linear inequalities, and therefore, they are not amenable to the Petri net-based methodologies discussed in Chapter 1. This effect was verified through techniques similar to those reported in Chapter 3, and it manifests another powerful attribute of the approach considered in this chapter with respect to other approaches reported in the current literature.

Closing this section on our computational experiments, we also report that these experiments were performed on a 2.66 GHz quad-core Intel Xeon 5430 processor with 6 MB of cache memory and 32 GB RAM; however, each job ran on a single core. The algorithms were encoded in C++, and they were compiled and linked by the

Table 5.3: A sample of our experimental results for the construction of non-parametric classifiers

$\|\xi\|$	$\|S_{rs}\|$	$\|S_{r\bar{s}}\|$	$\|\widehat{P(\hat{S}^b_{r\bar{s}})}\|$	$\|L_P\|$	# tab. entries	# dec. nodes	dec. diag. stor. reqs	compression ratio	comp. time (sec)
24	115766	28510	12	11	132	52	114	.86	1
32	163439	192381	131	23	3013	286	643	.21	7
40	42571	69016	188	30	5640	716	1580	.28	1
48	80343	691959	241	39	9399	818	1788	.19	24
52	1622861	2600349	257	37	9509	678	1473	.15	150
64	118470	1253425	612	54	33048	1895	4178	.13	55
70	10956	289962	338	58	19604	1245	2749	.14	12
77	15763	364985	1119	70	78330	2951	6472	.08	17
88	53080	1410311	1046	74	77404	3247	7069	.09	81
99	8425	413822	845	85	71825	3039	6682	.09	20

GNU g++ compiler under Unix. In all cases, the overall time necessary for the computation of the maximally permissive DAP and its representation through the corresponding n-ary decision diagram did not exceed the 600 sec; in fact, as revealed by the computational times reported in Table 5.3, in the majority of the considered cases the required computational time was less than 60 sec.[5]

5.6 Concluding remarks

This chapter has proposed a novel approach for the synthesis of maximally permissive DAP for sequential RAS in the form of non-parametric classifiers, and it has demonstrated the ability of these classifiers to provide effectively computable and practically implementable solutions for RAS with (very) large state spaces. In addition, the proposed method is complete, i.e., it is applicable to any RAS configuration from the considered RAS class. These capabilities arise from the ability to encode the information that is necessary for on-line implementation of the maximally permissive DAP in a very compact manner.

[5]We also note that a more detailed breakdown of these computational times reveals that most of the computational effort is expended on (i) the extraction of the set of boundary unsafe states, $S^b_{r\bar{s}}$, from the set of reachable unsafe states, $S_{r\bar{s}}$, and (ii) the identification of the minimal elements in $S^b_{r\bar{s}}$.

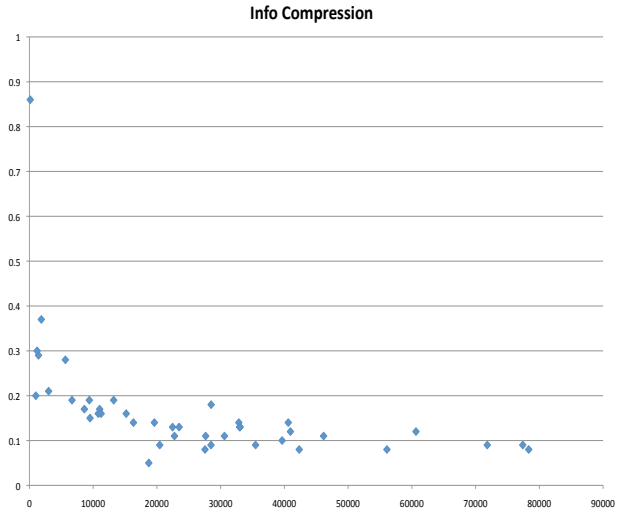


Figure 5.4: The information compression in the storage of the set $\widehat{P(\hat{S}^b_{r\bar{s}})}$ attained through the employment of the n-ary decision diagrams proposed in this chapter. The x-axis reports, for each considered RAS configuration, the product of the cardinality of $\widehat{P(\hat{S}^b_{r\bar{s}})}$ with the dimensionality of its elements; this product should be perceived as the total number of integer entries that are necessary to store $\widehat{P(\hat{S}^b_{r\bar{s}})}$ in a 2-dim array. The y-axis reports the storage capacity employed by the corresponding n-ary decision diagram as a percentage of the storage needs indicated in the x-axis.

Comparing the non-parametric classifiers introduced in this chapter to the parametric classifiers introduced in the previous chapter, we have the following comments:

- The parametric classifiers offer a more compact representation of the sought dichotomy. Thus, in general, the on-line deployment of the parametric classifiers is more efficient than that of the non-parametric ones, in terms of its memory requirements.

- In the parametric classifiers, the assessment of the state safety reduces to checking the satisfaction of a set of equations. On the other hand, in the non-parametric classifier, the assessment of the state safety requires the utilization

of graph-search procedures, which in the worst case, might visit all the nodes of the given graph. Hence, in general, the on-line computational overhead introduced by the parametric classifiers is smaller.

- Once the sets of states that are required for synthesizing a non-parametric classifier are available, constructing the classifier involves the application of graph compression procedures. On the other hand, once the corresponding sets of states required for synthesizing a parametric classifier are available, constructing the classifier involves solving a combinatorial optimization problem. Hence, in general, the off-line computations required for constructing the non-parametric classifiers are more benign[6].

- For the construction of the non-parametric classifiers, we need only to have the minimal reachable unsafe states. On the other hand, for the construction of the parametric classifiers, both the maximal reachable safe states and the minimal reachable unsafe states are required. In the last part of this thesis (Chapters 6 - 7), we shall leverage this advantage, offered by the non-parametric classifiers, by introducing an efficient algorithm that seeks to enumerate the minimal reachable unsafe states, while avoiding the complete enumeration of the state space.

[6]We remark that, in principle, it is possible to enhance the compactness of the non-parametric classifier, by permuting the components of the stored vectors, i.e., the components of the elements of $\widehat{P(\widehat{S}^b_{r\bar{s}})}$. However, the identification of an optimal permutation – i.e., a permutation that leads to a decision diagram with the smallest possible number of internal nodes – is an NP-complete problem [15]. Hence, this optimal permutation problem is typically addressed by heuristics that adapt, to the considered application context, ideas and techniques borrowed from the area of combinatorial optimization.

CHAPTER VI

EFFICIENT ENUMERATION OF THE MINIMAL REACHABLE UNSAFE STATES

6.1 Introduction

A substantial computational "bottleneck" in all the developments in the previous chapters is defined by the fact that they presuppose the availability of the enumerations of the RAS safe and unsafe subspaces. These enumerations are typically produced through the trimming of the FSA that models the RAS behavior, as described in Chapter 1. But it is well-known that the aforementioned FSA is of super-polynomial size w.r.t. the size of the more compact representations of the underlying RAS structure, and, in fact, it can grow prohibitively large for the purposes of the aforementioned computations, even for moderate RAS sizes. On the other hand, it should be evident from the previous chapters that what is really necessary for the construction of the sought classifiers is only a subset of the entire state space, which is quite smaller than the size of the entire state space, usually by many orders of magnitude.

Motivated by the above remarks, this chapter proposes a new algorithm that can support the enumeration of all the minimal reachable unsafe states, while avoiding the complete enumeration of the underlying state space. However the presented results are restricted to RAS classes that, in addition to the qualifications provided in Chapter 1 (c.f. Def. 1.1 and Assumptions 1.1 and 1.2) must further satisfy the following assumption:

Assumption 6.1 *In the RAS considered in this chapter, the data structure $\mathcal{G}_j$ that defines the sequential logic of process type J_j, $j = 1, \ldots, \zeta$, corresponds to a connected "*acyclic*" digraph $(\mathcal{V}_j, \mathcal{E}_j)$. Hence, all the process types possess no cyclic structure.*

The key idea for the proposed algorithm stems from the remark that, due to Assumption 6.1, unsafety is defined by unavoidable absorption into the system deadlocks. Hence, the unsafe states of interest can be retrieved by a localized computation that starts from the RAS deadlocks and "backtraces" the RAS dynamics until it hits the boundary between the safe and unsafe subspaces. In particular, our interest in minimal reachable unsafe states implies that we can focus this backtracing only to minimal reachable deadlocks. As a result, the proposed algorithm decomposes into a two-stage computation, with the first stage identifying the minimal reachable deadlock states, and the second stage performing the aforementioned backtracing process in order to identify the broader set of minimal reachable unsafe states. Together with the results of Chapter 5, the presented algorithm provides a powerful method for the deployment of the maximally permissive DAP even for RAS with extremely large state spaces.

In the light of the above discussion, the rest of the chapter is organized as follows: Section 6.2 presents the algorithm utilized for enumerating the minimal reachable deadlock states. Section 6.3 presents the algorithm that enumerates the minimal reachable unsafe states. Section 6.4 reports a series of computational experiments that demonstrate the extensive computational gains obtained by the proposed algorithm. Finally, Section 6.5 concludes the chapter.

6.2 Enumerating $\hat{S}_{rd}$

As pointed out in the introductory section, the first step towards the enumeration of the minimal reachable unsafe states is the enumeration of the minimal reachable deadlocks. This is the content of this section. First, we define some terms and notation that will be used throughout the rest of the chapter. Next, we proceed to describe the detailed flow of the proposed algorithm.

6.2.1 Preamble

Let $\mathbf{s}.R_i,\ i = 1,\ldots,\mu$, denote the total number of units from resource R_i that are allocated at state $\mathbf{s}$. Then, given an edge $e \in \mathcal{G}$, we shall say that e is "*blocked*" at state $\mathbf{s}$ *iff* $\mathbf{s}[e.src] > 0$, and $\exists R_k \in \mathcal{R}$ s.t. $\mathbf{s}.R_k + \mathcal{A}_{e.dst}[k] - \mathcal{A}_{e.src}[k] > C_k$. We shall say that edge $e \in \mathcal{G}$ is "*enabled*" at state $\mathbf{s}$ *iff* $\mathbf{s}[e.src] > 0$ and e is not blocked. Similarly, a processing stage q is blocked at state $\mathbf{s}$ *iff* all its outgoing edges are blocked, whereas it is enabled *iff* at least one of its outgoing edges is enabled. The set of enabled edges at $\mathbf{s}$ will be denoted by $g(\mathbf{s})$.

Definition 6.1 *Given a set of states X, define the state λ_X by $\lambda_X[q] \equiv max_{\mathbf{x}\in X}\mathbf{x}[q]$, $q = 1\ldots\xi$. In the sequel, we shall refer to λ_X as the* "combination" *of the states of X.*

By its definition, a minimal deadlock state $\mathbf{s}_d$ is a state at which all its processing stages with non-zero process content are blocked. Let $\{q_1,\ldots,q_t\}$ refer to this set of processing stages. Then, we have the following lemma:

Lemma 6.1 *A minimal deadlock state $\mathbf{s}_d$ with active processing stages $\{q_1,\ldots,q_t\}$ can be expressed as the combination of a set of minimal states $\{\mathbf{x}_1,\ldots,\mathbf{x}_t\}$ such that q_i is blocked at $\mathbf{x}_i$.*

Proof: Consider the vectors $\mathbf{x}_i,\ i = 1,\ldots,t$, that are obtained by starting from the deadlock state $\mathbf{s}_d$ and iteratively removing processes from this state, one at a time, until no further process can be removed without unblocking stage q_i. Then, clearly, each state $\mathbf{x}_i$ is a minimal state at which processing stage q_i is blocked. Furthermore, by the construction of $\{\mathbf{x}_i, i = 1,\ldots,t\}$, $\lambda_{\{\mathbf{x}_1,\ldots,\mathbf{x}_t\}} \preceq \mathbf{s}_d$, and $\lambda_{\{\mathbf{x}_1,\ldots,\mathbf{x}_t\}}$ is itself a deadlock state. But then, the minimality of $\mathbf{s}_d$ implies that $\lambda_{\{\mathbf{x}_1,\ldots,\mathbf{x}_t\}} = \mathbf{s}_d$. □

Let $\{e_{i1},\ldots,e_{i\mathcal{D}(q_i)}\}$ refer to the set of edges emanating from node q_i. Then, by an argument similar to that in the proof of Lemma 6.1, we can perceive each state $\mathbf{x}_i$

appearing in the statement of Lemma 6.1 as a combination of a set of minimal states $\{\mathbf{x}_{i1},\ldots,\mathbf{x}_{i\mathcal{D}(q_i)}\}$ such that e_{ij} is blocked at state $\mathbf{x}_{ij}$, i.e., $\mathbf{x}_i = \lambda_{\{\mathbf{x}_{i1},\ldots,\mathbf{x}_{i\mathcal{D}(q_i)}\}}$. Each $\mathbf{x}_{ij}$ is a state that has active processes at stage $e_{ij}.src$, and for some resource type R_k s.t. $\mathcal{A}_{e_{ij}.dst}[k] - \mathcal{A}_{e_{ij}.src}[k] > 0$, $\mathbf{x}_{ij}.R_k > C_k - \mathcal{A}_{e_{ij}.dst}[k] + \mathcal{A}_{e_{ij}.src}[k] \equiv l$. Hence, the minimal states that block e_{ij} through R_k can be obtained by enumerating all the minimal states that allocate $l+1,\ldots,C_k$ units of R_k (i.e., those minimal states for which $\mathbf{s}.R_k \in \{l+1,\ldots,C_k\}$).

Outline of the proposed algorithm: The proposed algorithm for the enumeration of the minimal reachable deadlocks is motivated by the analysis presented in the pervious paragraph, and it can be outlined as follows:

1. For each resource type R_k, and for each occupancy level l, $1 \leq l \leq C_k$, compute the set of minimal states that allocate l units of R_k; call it $MinStR[k][l]$.

2. Use the results obtained in Step 1 in order to compute, for each edge e, the set of minimal states at which e is blocked; call it $BlockEd[e]$.

3. Use the results obtained in Step 2 in order to compute, for each processing stage q, the set of minimal states at which q is blocked; call it $BlockPs[q]$.

4. Finally, enumerate the set of minimal deadlocks through the following recursive scheme that, for each processing stage q and each minimal state $\mathbf{s} \in BlockPs[q]$, does the following: It sets $\mathbf{p}_1 := \mathbf{s}$, and then searches for an enabled processing stage q' at $\mathbf{p}_1$. Next, it branches for each minimal state $\mathbf{x}$ at which q' is blocked (i.e., $\mathbf{x} \in BlockPs[q']$), combining such a state with $\mathbf{p}_1$ (i.e., it computes the combination $\lambda_{\{\mathbf{p}_1,\mathbf{x}\}}$). Let $\mathbf{p}_2$ be a (feasible) state generated at one of those branches; i.e., $\mathbf{p}_2 = \lambda_{\{\mathbf{p}_1,\mathbf{x}'\}}$, $\mathbf{x}' \in BlockPs[q']$. State $\mathbf{p}_2$ is processed in a similar manner with state $\mathbf{p}_1$ above, and the branching continues across all the generated paths of the resulting search graph until a deadlock state is reached on each path.

The rest of this section details the various steps in the above outline. Note that a process instance executing a terminal processing stage can immediately exit the system upon completion; hence terminal processing stages do not have any active process instances at any minimal deadlock state. Therefore, these stages will be ignored in the subsequent constructions.

6.2.2 Computing $MinStR[k][l]$

A minimal state $\mathbf{s}$ that allocates l units of resource type R_k may be either a unit vector state with $\mathbf{s}[q] = 1$ for some component $q \in \{1, \ldots, \xi\}$, $\mathbf{s}[q'] = 0$, $\forall q \neq q'$, and $\mathcal{A}_q[k] = l$, or a vector equal to $\mathbf{s}_1 + \mathbf{s}_2$ where $\mathbf{s}_1$ is a minimal state using j units of R_k and $\mathbf{s}_2$ is a minimal state using $l - j$ units of R_k. Based on the above remark, $MinStR[k][l]$ is initialized with the η_{kl} unit vector states corresponding to the stages that request l units of R_k. In particular, $MinStR[k][1]$ will contain only these η_{k1} unit vector states. Proceeding inductively for $l > 1$, and assuming that, $\forall j \leq \lfloor l/2 \rfloor$, $MinStR[k][j]$ has been already computed, we add each state in $MinStR[k][j]$ to each state in $MinStR[k][l-j]$, and insert the resultant states into $MinStR[k][l]$, provided that they satisfy the feasibility conditions of Eq. 1.1. The complete algorithm for computing $MinStR$ is depicted in Procedure 6.1.

Complexity analysis: To facilitate the complexity analysis of Procedure 6.1, define $\eta_k \equiv max_{l=1}^{C_k} \eta_{lk}$. Next, consider $MinStR[k][2]$. The list will have η_{k2} unit vector states, and $\mathcal{O}(\eta_{k1}^2)$ states formed by adding pairs of states in $MinStR[k][1]$. Hence, $MinStR[k][2]$ will have $\mathcal{O}(\eta_k^2)$ states. Similarly, $MinStR[k][3]$ will have η_{k3} unit vector states and $\mathcal{O}(\eta_{k1} \cdot (\eta_{k2} + \eta_{k1}^2)) = \mathcal{O}(\eta_{k2} \cdot \eta_{k1} + \eta_{k1}^3) = \mathcal{O}({\eta_k}^3)$ states formed by adding pairs of states from $MinStR[k][1]$ and $MinStR[k][2]$. By induction, we can see that $MinStR[k][l]$ will contain $\mathcal{O}({\eta_k}^l)$ states. Hence, the computation of the entire set of lists $MinStR[k][l]$, $l = 1 : C_k$, involves the construction and the feasibility assessment of $\mathcal{O}(\sum_{l=1}^{C_k} {\eta_k}^l) = \mathcal{O}({\eta_k}^{C_k})$ states. The construction of a new state results

Procedure 6.1 CompMinStR(k)

Input: a resource index k
Output: $MinStR[k][l],\ l = 1 : C_k$

```
1: for l = 1 : C_k do
2:    Initialize MinStR[k][l] with the corresponding η_kl unit vectors.
3:    for j = 1 : ⌊l/2⌋ do
4:       for s_1 ∈ MinStR[k][j] do
5:          for s_2 ∈ MinStR[k][l − j] do
6:             if Feasible(s_1 + s_2) then
7:                Insert (s_1 + s_2) into MinStR[k][l]
8:             end if
9:          end for
10:      end for
11:   end for
12: end for
13: return MinStR[k][l], l = 1 : C_k
```

from the addition of a pair of states, which is supported with a complexity of $\mathcal{O}(\xi)$. On the other hand, checking the feasibility of a given state has a complexity of $\mathcal{O}(\mu \cdot \xi)$. Hence, the overall complexity of Procedure 6.1 is $\mathcal{O}(\mu \cdot \xi \cdot {\eta_k}^{C_k})$.

6.2.3 Computing $BlockEd[e]$

According to the remarks that were provided in Section 6.2.1, the computation of this data structure can be organized as follows: For each resource R_k s.t. $\mathcal{A}_{e.dst}[k] - \mathcal{A}_{e.src}[k] > 0$, and for each occupancy level l s.t. $l > C_k - \mathcal{A}_{e.dst}[k] + \mathcal{A}_{e.src}[k]$, we insert all the states from $MinStR[k][l]$ into $BlockEd[e]$ after adding one process at $e.src$, if needed; in this last case, the resulting state must also be checked for feasibility. The complete algorithm supporting this computation is depicted in Procedure 6.2.

Complexity analysis: As established in the complexity analysis of Procedure 6.1, $MinStR[k][l]$ contains $\mathcal{O}({\eta_k}^l)$ states. Hence, for a given k, the "For" loop at Lines 4-14 is executed $\mathcal{O}({\eta_k}^1 + {\eta_k}^2 + \ldots + {\eta_k}^{C_k}) = \mathcal{O}({\eta_k}^{C_k})$ times. But then, the overall complexity of Procedure 6.2 is $\mathcal{O}(\mu \cdot \xi \cdot \sum_{k=1}^{\mu} {\eta_k}^{C_k})$, where $\mathcal{O}(\mu \cdot \xi)$ is the complexity of checking the feasibility of a state. Also, the number of states in $BlockEd[e]$ is bounded above by $\mathcal{O}(\sum_{k=1}^{\mu} {\eta_k}^{C_k})$. Finally, to simplify the notation, in the following

Procedure 6.2 CompBlockEd(e)

Input: an edge e from the "union" process graph $\mathcal{G}$
Output: $BlockEd[e]$

```
1: for k := 1 → μ do
2:   if A_{e.dst}[k] − A_{e.src}[k] > 0 then
3:     for l = C_k − A_{e.dst}[k] + A_{e.src}[k] + 1 → C_k do
4:       for s ∈ MinStR[k][l] do
5:         s' ← s
6:         if s'[e.src] = 0 then
7:           s'[e.src] ← 1
8:           if Feasible(s') then
9:             Insert s' into BlockEd[e]
10:          end if
11:        else
12:          Insert s' into BlockEd[e]
13:        end if
14:      end for
15:    end for
16:  end if
17: end for
18: return BlockEd[e]
```

we shall denote $\sum_{k=1}^{\mu} \eta_k{}^{C_k}$ by ρ.

6.2.4 Computing $BlockPs[q]$

Let $\{e_1^q, \ldots, e_{\mathcal{D}(q)}^q\}$ be the set of edges emanating from q. Then, $BlockPs[q]$ is computed by taking all the feasible combinations of states from $BlockEd[e_1^q] \times BlockEd[e_2^q] \times \ldots BlockEd[e_{\mathcal{D}(q)}^q]$, while eliminating those combinations that result in non-minimal elements. The complete algorithm for this enumeration is depicted in Procedure 6.3. The procedure proceeds in a recursive manner, starting with an input of the considered processing stage q and the values $i = 0$, $\mathbf{s} = \mathbf{0}$, and $BlockPs[q] = \emptyset$ for its remaining parameters. The main part of the pursued computation is supported by Lines 8-13, that combine each (partially) generated state $\mathbf{s}$ with each minimal state that blocks e_i^q, check the feasibility of the resulting state, and for each feasible combination, proceed recursively to block edge e_{i+1}^q. On the other hand, the condition at Line 6 addresses the case where edge e_i^q is already blocked at the processed state $\mathbf{s}$.

Procedure 6.3 CompBlockPs(q, i, $\mathbf{s}$, $BlockPs[q]$)

Input: processing stage q,
index i, * set to 0 in the first call *\
a partially constructed state $\mathbf{s}$, * set to $\mathbf{0}$ in the first call *\
the partially constructed output state set $BlockPs[q]$ * set to $\emptyset$ in the first call *\
Output: $BlockPs[q]$

1: **if** $i > \mathcal{D}(q)$ **then**
2: Insert $\mathbf{s}$ into $BlockPs[q]$;
3: **return** $BlockPs[q]$
4: **end if**
5: **if** $\exists \mathbf{x} \in BlockEd[e_i^q]$ s.t. $\mathbf{s} \succeq \mathbf{x}$ **then**
6: $BlockPs[q] \leftarrow CompBlockPs(q, i+1, \mathbf{s}, BlockPs[q])$;
7: **else**
8: **for** $\mathbf{x} \in BlockEd[e_i^q]$ **do**
9: $\mathbf{s}' \leftarrow \lambda_{\mathbf{x},\mathbf{s}}$
10: **if** $Feasible(\mathbf{s}')$ **then**
11: $BlockPs[q] \leftarrow CompBlockPs(q, i+1, \mathbf{s}', BlockPs[q])$;
12: **end if**
13: **end for**
14: **end if**
15: Remove non-minimal states from $BlockPs[q]$
16: **return** $BlockPs[q]$

Lines 1-4 define the terminal condition of the recursion, where all the outgoing edges have been blocked at the given state $\mathbf{s}$. Hence, $\mathbf{s}$ is added to $BlockPs[q]$ at Line 2. Finally, the non-minimal states are removed at Line 15.

Complexity analysis: We can see that each recursive path is of depth $\mathcal{D}(q)$, and that we have no more than $\prod_{i=1}^{\mathcal{D}(q)} |BlockEd[e_i^q]|$ possible state combinations from $BlockEd$. On the other hand, as it can be deduced from the complexity analysis of Procedures 6.1–6.2, Lines 9-10 are supported with complexity $\mathcal{O}(\mu \cdot \xi)$, and the number of states in $BlockEd[e_i^q]$ is bounded above by $\mathcal{O}(\rho)$. Therefore, Lines 1-14 are supported with a complexity of $\mathcal{O}(\mu \cdot \xi \cdot \rho^{\mathcal{D}(q)})$. Also, the number of states in $BlockPs[q]$ is bounded above by $\mathcal{O}(\rho^{\mathcal{D}(q)})$. Finally, Line 15 boils down to pairwise comparisons of the generated states. Hence, its complexity is $\mathcal{O}(\xi \cdot \rho^{2 \cdot \mathcal{D}(q)})$. Therefore, the overall complexity of Procedure 6.3 is $\mathcal{O}(\mu \cdot \xi \cdot \rho^{\mathcal{D}(q)} + \xi \cdot \rho^{2 \cdot \mathcal{D}(q)}) = \mathcal{O}(\xi \cdot \rho^{2 \cdot \mathcal{D}(q)})$.

6.2.5 Enumerating the minimal reachable deadlock states

The complete algorithm for enumerating the minimal reachable deadlock states is depicted in Procedure 6.4. Lines 2-10 involve the computation of the lists $MinStR$, $BlockEd$, and $BlockPs$. For each processing stage q, all the minimal deadlock states at which q has non-zero processes are enumerated by Lines 11-28. In particular, for a given processing stage q, we start by inserting into the list *workingQueue* each minimal state at which q is blocked. In the "While" loop of Lines 14-26, we extract every state $\mathbf{p}$ from this queue and examine $\mathbf{p}$ for enabled processing stages. If $\mathbf{p}$ has no enabled processing stages, the function $getAnEnabledProcStg$ at Line 16 returns a value of 0; hence, it is inferred that $\mathbf{p}$ is a deadlock state at which q has non-zero processes, and it is inserted into the hash table $deadlockHT$ (c.f. Line 18). Otherwise, $getAnEnabledProcStg$ returns an enabled processing stage q^*. In this case, Lines 20-24 generate every feasible combination of state $\mathbf{p}$ with the minimal states blocking q^* and add them to *workingQueue*. *workingQueue* becomes empty when all the deadlock states at which q has non-zero processes are enumerated on all the paths of the obtained search graph. Finally, Line 29 removes the non-minimal and unreachable deadlock states from $deadlockHT$.

Complexity analysis: We start with the complexity analysis of Lines 12-27. By an argument similar to that provided in the complexity analysis for Procedure 6.3, it is easy to see that there exist no more than $\prod_{i=1}^{q} |BlockPs[q]|$ possible state combinations forming deadlock states. Based on the analysis of Procedure 6.3, the number of states in $BlockPs[q]$ is bounded above by $\mathcal{O}(\rho^{\mathcal{D}(q)})$. On the other hand, combining a pair of states and checking the feasibility of the resultant state (Line 21) have a combined complexity of $\mathcal{O}(\mu \cdot \xi)$, whereas getting an enabled processing stage (Line 16) has a complexity of $\mathcal{O}(\mu \cdot |\mathcal{E}|)$. Hence, the overall complexity of Lines 12-27 of Procedure 6.4 is $\mathcal{O}(\mu \cdot |\mathcal{E}| \cdot \rho^{\sum_{q=1}^{\xi} \mathcal{D}(q)}) = \mathcal{O}(\mu \cdot |\mathcal{E}| \cdot \rho^{|\mathcal{E}|})$. Similarly, it can be seen that

Procedure 6.4 EnumMinReachDeadlocks(Φ)

Input: A RAS instance Φ

Output: the list *deadlockHT* containing all the reachable minimal deadlocks of Φ

```
1: deadlockHT ← ∅
2: for k = 1 → μ do
3:    MinStR[k] ← CompMinStR(k)
4: end for
5: for all e ∈ E do
6:    BlockEd[e] ← CompBlockEd(e);
7: end for
8: for q = 1 → ξ do
9:    BlockPs[q] ← CompBlockPs(q, 0, 0, ∅)
10: end for
11: for q = 1 : ξ do
12:    for s ∈ BlockPs[q] do
13:       workingQueue ← s;
14:       while workingQueue ≠ ∅ do
15:          p ← dequeue(workingQueue)
16:          q* ← getAnEnabledProcStg(p)
17:          if q* = 0 then
18:             Insert p into deadlockHT
19:          else
20:             for all x ∈ BlockPs[q*] do
21:                if Feasible(λ_{x,p}) then
22:                   Insert λ_{x,p} into workingQueue
23:                end if
24:             end for
25:          end if
26:       end while
27:    end for
28: end for
29: Remove non-minimal states and unreachable states from deadlockHT
30: return deadlockHT
```

an upper bound for the number of states in *deadlockHT* is $\mathcal{O}(\rho^{|\mathcal{E}|})$. Thus, the pairwise comparison needed to eliminate the non-minimal deadlock states at Line 29 can be supported by $\mathcal{O}(\rho^{2\cdot|\mathcal{E}|}\cdot\xi)$ operations. On the other hand, the reachability assessment performed at Line 29 is implemented using a depth-first search supported with hash tables to mark visited states. Hence, a worst-case bound for this part of the

computation is given by $\mathcal{O}(|S_r|)$, but practically, due to the small process content of the assessed states, the run time of this step is very small.[1]

Based on the complexity analysis of Procedure 6.4 discussed above, and the analysis of its subroutines that were presented in the previous subsections, we can summarize the overall complexity of Procedure 6.4 as follows: (i) Lines 2-4 are of complexity $\mathcal{O}(\mu \cdot \xi \cdot \rho)$. (ii) Lines 5-7 are of complexity $\mathcal{O}(|\mathcal{E}| \cdot \xi \cdot \mu \cdot \rho)$. (iii) Lines 8-10 are of complexity $\mathcal{O}(\xi \cdot \sum_{q=1}^{\xi} \rho^{2 \cdot \mathcal{D}(q)})$. (iv) Lines 11-28 are of complexity $\mathcal{O}(\xi \cdot \mu \cdot |\mathcal{E}| \cdot \rho^{|\mathcal{E}|})$. (v) Line 29 is of complexity $\mathcal{O}(\rho^{2 \cdot |\mathcal{E}|} \cdot \xi + |S_r|)$.

Therefore, the overall complexity is $\mathcal{O}((\mu \cdot \xi \cdot \rho + |\mathcal{E}| \cdot \xi \cdot \mu \cdot \rho + \xi \cdot \sum_{q=1}^{\xi} \rho^{2 \cdot \mathcal{D}(q)} + \xi \cdot \mu \cdot |\mathcal{E}| \cdot \rho^{|\mathcal{E}|} + \xi \cdot \rho^{2 \cdot |\mathcal{E}|} + |S_r|) \approx \mathcal{O}(\xi \cdot \rho^{2 \cdot |\mathcal{E}|} + |S_r|)$. Let $\eta \equiv max_{k=1}^{\mu} \eta_k$, and $C = max_{k=1}^{\mu} C_k$. Hence, the overall complexity can be rewritten as $\mathcal{O}(\xi \cdot (\sum_{i=1}^{\mu} \eta_k^{C_k})^{2 \cdot |\mathcal{E}|} + |S_r|) = \mathcal{O}(\xi \cdot \mu^{2 \cdot |\mathcal{E}|} \cdot \eta^{2 \cdot C \cdot |\mathcal{E}|} + |S_r|)$. From this last expression we can conclude that the algorithm complexity is most sensitive to the capacity of the resource types and to the number of the distinct event types that take place in the underlying RAS.

The next theorem establishes the correctness of Procedure 6.4.

Theorem 6.1 *Procedure 6.4 enumerates all the minimal reachable deadlock states.*

Proof: Let $\mathbf{s}_d$ be an arbitrary minimal deadlock state and $\{q_1, \ldots, q_t\}$ be the set of processing stages that have active processes at $\mathbf{s}_d$. Then, according to Lemma 6.1, there exists a set of states $\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$ such that $\mathbf{x}_j \in BlockPs[q_j]$, $\mathbf{x}_j \preceq \mathbf{s}_d$, and $\mathbf{s}_d = \lambda_{\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}}$. $\mathbf{x}_1$ will be picked by Line 12. Without loss of generality (w.l.o.g.), assume that $q_2 = getAnEnabledProcStg(\mathbf{x}_1)$. Then, Line 22 implies that the state $\mathbf{p}_2 = \lambda_{\{\mathbf{x}_1, \mathbf{x}_2\}}$ is inserted into *workingQueue*; hence, it will be eventually extracted at

[1] This claim is further substantiated by the computational experiments that are presented in Section 6.4. Also, we notice that it is possible to skip the elimination of the reachable unsafe states in the construction of the list *deadlockHT*, without compromising the correctness of the resulting implementation of the maximally permissive DAP. However, the presence of the unreachable deadlock states in *deadlockHT* would have an adversarial impact on the complexity of the computation of the set $\hat{S}_{r\bar{s}}$ that is discussed in the next section, that is much more severe than the computational cost of their removal from that list.

Line 15. Repeating the same argument, assume that $q_3 = getAnEnabledProcStg(\mathbf{p}_2)$; then, we will have the state $\mathbf{p}_3 = \lambda_{\{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3\}}$ inserted into $workingQueue$. Let $t' \leq t$ be the last processing stage in this tracing sequence. Thus, $\mathbf{p}_{t'} = \lambda_{\{\mathbf{x}_1,\ldots,\mathbf{x}_{t'}\}}$ is a deadlock state. But, $\mathbf{p}_{t'} = \lambda_{\{\mathbf{x}_1,\ldots,\mathbf{x}_{t'}\}} \preceq \lambda_{\{\mathbf{x}_1,\ldots,\mathbf{x}_t\}} = \mathbf{s}_d$. Therefore, by the minimality of $\mathbf{s}_d$, it must be that $\mathbf{s}_d = \mathbf{p}_{t'}$. Hence, $\mathbf{s}_d$ is enumerated. □

Example 6.1 We shall demonstrate the enumeration of the minimal deadlock states through the algorithms introduced above, using the RAS configuration depicted in Table 6.1. The considered RAS has five resource types, $\{R_1, \ldots, R_5\}$. Resources R_1, R_2 and R_5 have a capacity of one unit, whereas the capacity of R_3 and R_4 is equal to two. The considered RAS also has three process types $\{J_1, J_2, J_3\}$, with J_2 and J_3 having a simple linear structure. On the other hand, J_1 presents routing flexibility. More specifically, a job at the second processing stage Ξ_{12} can advance to Ξ_{13} (acquiring R_3), or to Ξ_{14} (acquiring R_4). For representational economy, in the subsequent discussion a state will be represented by the multi-set of the processing stages with non-zero process content in it.

The arrays $MinStR$, $BlockEd$, and $BlockPs$ computed for this example are respectively depicted in Tables 6.2, 6.3, and 6.4. As previously mentioned, terminal stages Ξ_{15}, Ξ_{22}, Ξ_{32} have been discarded from consideration. We note from Table 6.4 that $BlockPs[\Xi_{13}]$ and $BlockPs[\Xi_{14}]$ are empty, and hence, these processing stages cannot be involved in a minimal deadlock. This is because the advancement of a process instance W_1 that executes any of these two stages can be blocked only by a process instance W_2 that holds the single unit of capacity of resource R_5, which is requested for the next processing stage of W_1. But resource R_5 is supporting exclusively processing stage Ξ_{15}, which is a terminal stage. Thus, W_2 will eventually finish, allowing W_1 to advance.

The above remarks further imply that, during the application of Procedure 6.4, states $\mathbf{ps}_2$–$\mathbf{ps}_5$ and $\mathbf{ps}_7$–$\mathbf{ps}_{10}$ cannot be involved in the formation of a deadlock,

Table 6.1: The RAS considered in Example 6.1

Resource Types:	$\{R_1, \ldots, R_5\}$
Resource Capacities:	$C_1 = C_2 = C_5 = 1$, $C_3 = C_4 = 2$
Process Type 1:	$R_1 \rightarrow R_2 \rightarrow (R_3 \text{ or } R_4) \rightarrow R_5$
Process Type 2:	$R_3 \rightarrow R_1$
Process Type 3:	$R_4 \rightarrow R_1$

Table 6.2: The array $MinStR$ for Example 6.1

$MinStR[1][1]$	$\{\Xi_{11}\}$
$MinStR[2][1]$	$\{\Xi_{12}\}$
$MinStR[3][1]$	$\{\Xi_{13}\}$, $\{\Xi_{21}\}$
$MinStR[3][2]$	$\{2\Xi_{13}\}$, $\{2\Xi_{21}\}$, $\{\Xi_{13}, \Xi_{21}\}$
$MinStR[4][1]$	$\{\Xi_{14}\}$, $\{\Xi_{31}\}$
$MinStR[4][2]$	$\{2\Xi_{14}\}$, $\{2\Xi_{31}\}$, $\{\Xi_{14}, \Xi_{31}\}$
$MinStR[5][1]$	$\emptyset$

Table 6.3: The array $BlockEd$ for Example 6.1

$BlockEd[(\Xi_{11} \rightarrow \Xi_{12})]$	$\mathbf{es}_1 = \{\Xi_{11}, \Xi_{12}\}$
$BlockEd[(\Xi_{12} \rightarrow \Xi_{13})]$	$\mathbf{es}_2 = \{\Xi_{12}, 2\Xi_{13}\}$, $\mathbf{es}_3 = \{\Xi_{12}, 2\Xi_{21}\}$, $\mathbf{es}_4 = \{\Xi_{12}, \Xi_{13}, \Xi_{21}\}$
$BlockEd[(\Xi_{12} \rightarrow \Xi_{14})]$	$\mathbf{es}_5 = \{\Xi_{12}, 2\Xi_{14}\}$, $\mathbf{es}_6 = \{\Xi_{12}, 2\Xi_{31}\}$, $\mathbf{es}_7 = \{\Xi_{12}, \Xi_{14}, \Xi_{31}\}$
$BlockEd[(\Xi_{13} \rightarrow \Xi_{15})]$	$\emptyset$
$BlockEd[(\Xi_{14} \rightarrow \Xi_{15})]$	$\emptyset$
$BlockEd[(\Xi_{21} \rightarrow \Xi_{22})]$	$\mathbf{es}_8 = \{\Xi_{21}, \Xi_{11}\}$
$BlockEd[(\Xi_{31} \rightarrow \Xi_{32})]$	$\mathbf{es}_9 = \{\Xi_{31}, \Xi_{11}\}$

because each of them contains active process instances in stages Ξ_{13} or Ξ_{14}, and as explained above, none of these stages can be blocked in a permanent manner. On the other hand, consider the iteration of Procedure 6.4 that starts from stage Ξ_{11} and state $\mathbf{ps}_1 = \{\Xi_{11}, \Xi_{12}\}$. To block Ξ_{12}, we can try combine state $\mathbf{ps}_1$ with any of the states $\mathbf{ps}_2$–$\mathbf{ps}_{10}$ (c.f. Table 6.4). However, based on the previous remarks, only the combination with state $\mathbf{ps}_6$ might lead to a deadlock. Indeed, combining $\mathbf{ps}_1$ and $\mathbf{ps}_6$ results in the state $\{\Xi_{11}, \Xi_{12}, 2\Xi_{21}, 2\Xi_{31}\}$, which is a state at which all the active processing stages are blocked; hence, it is a deadlock state. Continuing the application of the algorithm does not yield any other deadlock state. □

Table 6.4: The array $BlockPs$ for Example 6.1

$BlockPs[\Xi_{11}]$	$\mathbf{ps}_1 = \{\Xi_{11}, \Xi_{12}\}$
$BlockPs[\Xi_{12}]$	$\mathbf{ps}_2 = \{\Xi_{12}, 2\Xi_{13}, 2\Xi_{14}\}$, $\mathbf{ps}_3 = \{\Xi_{12}, 2\Xi_{13}, 2\Xi_{31}\}$, $\mathbf{ps}_4 = \{\Xi_{12}, 2\Xi_{13}, \Xi_{14}, \Xi_{31}\}$, $\mathbf{ps}_5 = \{\Xi_{12}, 2\Xi_{14}, 2\Xi_{21}\}$, $\mathbf{ps}_6 = \{\Xi_{12}, 2\Xi_{21}, 2\Xi_{31}\}$, $\mathbf{ps}_7 = \{\Xi_{12}, \Xi_{14}, 2\Xi_{21}, \Xi_{31}\}$, $\mathbf{ps}_8 = \{\Xi_{12}, \Xi_{13}, 2\Xi_{14}, \Xi_{21}\}$, $\mathbf{ps}_9 = \{\Xi_{12}, \Xi_{13}, \Xi_{21}, 2\Xi_{31}\}$, $\mathbf{ps}_{10} = \{\Xi_{12}, \Xi_{13}, \Xi_{14}, \Xi_{21}, \Xi_{31}\}$
$BlockPs[\Xi_{13}]$	$\emptyset$
$BlockPs[\Xi_{14}]$	$\emptyset$
$BlockPs[\Xi_{21}]$	$\mathbf{ps}_{11} = \{\Xi_{11}, \Xi_{21}\}$
$BlockPs[\Xi_{31}]$	$\mathbf{ps}_{12} = \{\Xi_{11}, \Xi_{31}\}$

6.3 Enumerating $\hat{S}_{r\bar{s}}$

In this section, we provide an algorithm that enumerates the subspace $\hat{S}_{r\bar{s}}$ without enumerating the entire reachable state space. We proceed as follows: First, we introduce all the necessary definitions for the description of the algorithm. Next, we introduce the algorithm itself. Finally, we prove the correctness of the algorithm.

6.3.1 Preamble

Given a minimal deadlock-free unsafe state $\mathbf{u}$, we notice the following: (i) No unloading event is enabled at $\mathbf{u}$, since otherwise $\mathbf{u}$ would be non-minimal. (ii) The unsafety of $\mathbf{u}$ is a consequence of its current process content and it does not require the loading of any new processes in order to manifest itself. (iii) The advancement of any unblocked process at $\mathbf{u}$ leads to another unsafe state; however, this new unsafe state can be minimal or non-minimal. The following definition characterizes further the dynamics that result from the advancement of unblocked processes in a minimal unsafe state.

Definition 6.2 *Given a minimal unsafe state* $\mathbf{u}$ *such that* $g(\mathbf{u}) = \{e_1, \ldots, e_K\}$*, let* $\mathbf{h}_1, \ldots, \mathbf{h}_K$ *be the respective states that result from executing events* $e_1, \ldots, e_K$ *at* $\mathbf{u}$*. Then,* $\forall i = 1 : K$*,* $nextMin(\mathbf{u}, e_i) \equiv \{\mathbf{z}_{i1}, \ldots, \mathbf{z}_{iw(i)}\}$ *where* $\forall j = 1 : w(i)$*,* $\mathbf{z}_{ij} \preceq \mathbf{h}_i$ *is a minimal unsafe state. We also set* $nextMin(\mathbf{u}) \equiv \bigcup_{i=1}^{K} nextMin(\mathbf{u}, e_i)$*. Finally, we denote by* $\mathbf{s}_{ij}$ *the result of backtracing* e_i *at* $\mathbf{z}_{ij}$*.*

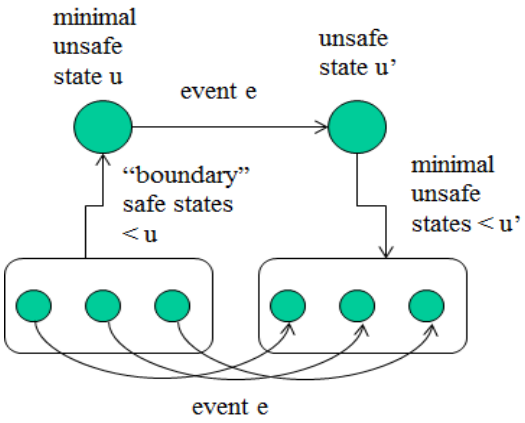


Figure 6.1: A schematic diagram of the RAS transitional structure that is leveraged by the proposed algorithm.

It is easy to see that if $\mathbf{h}_i$, in the above definition, is a minimal unsafe state, then $w(i) = 1$, $\mathbf{z}_{i1} = \mathbf{h}_i$, $\mathbf{s}_{i1} = \mathbf{u}$. Otherwise, to show that $\mathbf{s}_{ij}$ is well-defined, it suffices to show that: (i) $\mathbf{z}_{ij}[e_i.dst] = \mathbf{h}_i[e_i.dst]$, and (ii) state $\mathbf{s}_{ij}$ is a feasible state according to Eq. 1.1. To establish item (i), first notice that $e_i.dst$ is the unique entry for which $\mathbf{h}_i$ is greater than $\mathbf{u}$. Hence, if item (i) was not true, then $\mathbf{z}_{ij} \prec \mathbf{u}$, a result that violates the minimality of $\mathbf{u}$. On the other hand, item (ii) is established by the fact that $\mathbf{z}_{ij} \prec \mathbf{h}_i$. It can also be seen that if $\mathbf{z}_{ij} \prec \mathbf{h}_i$, then $\mathbf{s}_{ij} \prec \mathbf{u}$. Combined with the minimality of $\mathbf{u}$ as an unsafe state, this last result implies that $\mathbf{s}_{ij}$ is a safe state in this case. The structure revealed by Definition 6.2 and the above discussion is depicted schematically in Figure 6.1.

As explained in the introductory section, the algorithm proposed in this work seeks to enumerate all the minimal reachable unsafe states starting from the minimal reachable deadlocks, and tracing backwards the dynamics that are described in Definition 6.2. This reconstructive process can be described as follows: Let us characterize a safe state $\mathbf{a}$ as a "*boundary safe*" state *iff* it is one-step away from reaching some unsafe state. During the course of its execution, the proposed algorithm generates, both, unsafe and safe states. The generated safe states are all boundary safe

states, and they are used as "stepping stones" to reach further parts of the unsafe state space. More specifically, the proposed algorithm employs three different mechanisms to generate states in its exploration process: (i) backtracing from an unsafe state; (ii) combining two boundary safe states (i.e., taking the maximum number of processes at each processing stage); and (iii) adding some processes to a boundary safe state to make it unsafe. The first two mechanisms can return, both, safe and unsafe states, whereas the last mechanism returns only unsafe states. In the case of the first two mechanisms, once a state $\mathbf{a}$ has been generated, its potential unsafety will be identified by running upon it a search-type algorithm that assesses the state co-reachability w.r.t. the target state $\mathbf{s}_0$. If $\mathbf{a}$ is found to be unsafe, it is also tested for non-minimality w.r.t. the previously generated unsafe states; if it is minimal, it is backtraced to generate its immediate predecessors, and then it is saved. On the other hand, if the generated state $\mathbf{a}$ is safe, then, it is further characterized by an additional attribute that is computed upon its generation; this attribute will be denoted by $\tau_{\mathbf{a}}$ and it constitutes a set of edges that emanate from state $\mathbf{a}$ and have been identified as leading to unsafe states. Also, in the sequel, we shall denote by $U(\mathbf{a})$ the set of the unsafe states that are reached from $\mathbf{a}$ through the edges in $\tau_{\mathbf{a}}$. The detailed algorithm for the computation of the set $\tau_{\mathbf{a}}$ depends on the mechanism that generated safe state $\mathbf{a}$, and it can be described as follows:

- If $\mathbf{a}$ was generated by tracing back upon edge e from the unsafe state $\mathbf{u}$, then, obviously, firing e at $\mathbf{a}$ leads to unsafety. Hence, in this case, the algorithm sets $\tau_{\mathbf{a}} = \{e\}$.

- If $\mathbf{a}$ was generated by combining two previously generated boundary safe states $\mathbf{a}_1$ and $\mathbf{a}_2$ (i.e., $\mathbf{a} = \lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}$), it is easy to see that firing any enabled transition among $\tau_{\mathbf{a}_1} \cup \tau_{\mathbf{a}_2}$ at state $\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}$ will lead to a state that dominates a state in $U(\mathbf{a}_1) \cup U(\mathbf{a}_2)$; hence, to an unsafe state. Thus, in this case, $\tau_{\mathbf{a}} = (\tau_{\mathbf{a}_1} \cup \tau_{\mathbf{a}_2}) \cap$

$g(\mathbf{a})$.

Furthermore, it is possible that a boundary safe state $\mathbf{a}$ will be generated more than once in the execution of the proposed algorithm. In particular, it might happen that $\mathbf{a'}_1 = \mathbf{a'}_2$, but $\tau_{\mathbf{a'}_1} \neq \tau_{\mathbf{a'}_2}$. This will happen if $\mathbf{a'}_1$ and $\mathbf{a'}_2$ are generated by different mechanisms, by backtracing from different unsafe states, or by combining different pairs of boundary safe states. Assume w.l.o.g. that $\mathbf{a'}_1$ was generated first in the course of the algorithm execution. Then, $\mathbf{a'}_2$ will be discarded upon its generation, but $\tau_{\mathbf{a'}_1}$ will be updated to $\tau_{\mathbf{a'}_1} := \tau_{\mathbf{a'}_1} \cup \tau_{\mathbf{a'}_2}$.

The rationale for the three state-generation mechanisms that were described above in the overall search process for minimal unsafe states, can be explained as follows: The first mechanism is the primary backtracing mechanism employed by the proposed algorithm, and therefore, its role is self-explanatory. On the other hand, in order to explain the role of the second and the third mechanisms, we remind the reader that Definition 6.2 implies that a boundary safe state $\mathbf{a}$ might be dominated by another minimal unsafe state leading to unsafe states that dominate some state(s) in $U(\mathbf{a})$ (c.f. also Figure 6.1); these two state-generation mechanisms enable the proposed algorithm to reach these additional minimal unsafe states. More specifically, by applying the second mechanism on any pair of boundary safe states $\mathbf{a}_1$ and $\mathbf{a}_2$, we obtain the state $\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}$ that dominates both $\mathbf{a}_1$ and $\mathbf{a}_2$ w.r.t. the partial state order that is established by '$\preceq'$. This domination further implies that $g(\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}) \subseteq g(\mathbf{a}_1) \cup g(\mathbf{a}_2)$. Furthermore, if $\tau_{\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}} \equiv \{\tau_{\mathbf{a}_1} \cup \tau_{\mathbf{a}_2}\} \cap g(\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}) \neq \emptyset$, the aforementioned domination also implies that $g(\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}})$ contains transitions to states that dominate unsafe states and, therefore, they are themselves unsafe. Hence, the constructed state $\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}$ is either an unsafe state, or if it is safe, it remains boundary. In the former case, the mechanism has succeeded in its objective of reaching a new part of the unsafe region, as described above. In the second case, the mechanism provides another boundary safe state that can be used for the generation of new unsafe states through

the second and the third mechanism. Finally, when using the third mechanism, we seek to add some processes to a boundary safe state $\mathbf{a}$, in order to obtain a state $\mathbf{y}$ such that $g(\mathbf{y}) \subseteq \tau_{\mathbf{a}}$. Thus, any enabled transition at $\mathbf{y}$ leads to a state that dominates a state in $U(\mathbf{a})$; hence, to an unsafe state. Therefore, $\mathbf{y}$ is also unsafe.

The following definitions provide a more formal characterization for the second and the third mechanisms.

Definition 6.3 *Consider a pair of boundary safe states* $\mathbf{a}_1$ *and* $\mathbf{a}_2$*. The pair* $(\mathbf{a}_1, \mathbf{a}_2)$ *is "*combinable*"* iff *(i)* $\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}$ *satisfies Equation 1.1, and (ii)* $\tau_{\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}} \equiv \{\tau_{\mathbf{a}_1} \cup \tau_{\mathbf{a}_2}\} \cap g(\lambda_{\{\mathbf{a}_1,\mathbf{a}_2\}}) \neq \emptyset$.

Definition 6.4 *Given a boundary safe state* $\mathbf{a}$*, define the set of states* $Confine(\mathbf{a},\ \tau_{\mathbf{a}})$ *as follows:* $\mathbf{x}' \in\ Confine(\mathbf{a},\ \tau_{\mathbf{a}})$ iff *(i)* $\mathbf{x}' \succ \mathbf{a}$*, (ii)* $g(\mathbf{x}') \subseteq \tau_{\mathbf{a}}$*, (iii)* $\nexists \mathbf{y} \prec \mathbf{x}'$ *that satisfies (i) and (ii).*

Condition (iii) in the above definition eliminates non-minimal unsafe states. The next proposition shows that any state in $Confine(\mathbf{a},\ \tau_{\mathbf{a}})$ is an unsafe state.

Proposition 6.1 *If* $\mathbf{x}' \in\ Confine(\mathbf{a},\ \tau_{\mathbf{a}})$*, then* $\mathbf{x}'$ *is an unsafe state.*

Proof: Let $t_1 \in g(\mathbf{x}')$ be a transition. Definition 6.4 implies that $t_1 \in \tau_{\mathbf{a}}$. Hence, firing t_1 at $\mathbf{a}$ leads to an unsafe state $\mathbf{u}_1$. By Definition 6.4 again, $\mathbf{x}' \succ \mathbf{a}$. Therefore, firing t_1 at $\mathbf{x}'$ leads to a state that dominates $\mathbf{u}_1$, and therefore, to an unsafe state. Since $g(\mathbf{x}') \subseteq \tau_{\mathbf{a}}$, all enabled transitions at $\mathbf{x}'$ lead to unsafety. Hence, $\mathbf{x}'$ is an unsafe state. □

6.3.2 The proposed algorithm

We start the presentation of the proposed algorithm by introducing a subroutine for removing the non-minimal unsafe states that are generated during the course of the execution of the algorithm. Procedure 6.5 is invoked to insert the states in $\mathbf{U}$ into Q_1, while removing any non-minimal state vectors from $Q_1 \cup Q_2$. In particular, a state

Procedure 6.5 Insert_Non_Min(U, Q_1, Q_2)

Input: U, Q_1, Q_2
Output: Q_1, Q_2

1: $Q^* \leftarrow \emptyset$
2: **for all** $\mathbf{u} \in U$ **do**
3: $\quad \forall \mathbf{x} \in Q_1 \cup Q_2$ s.t. $\mathbf{x} \succ \mathbf{u}$, remove $\mathbf{x}$;
4: $\quad$ **if** $\nexists \mathbf{x} \in Q_1 \cup Q_2$ s.t. $\mathbf{x} \prec \mathbf{u}$ **then**
5: $\quad\quad$ Insert $\mathbf{u}$ into Q^*
6: $\quad$ **end if**
7: **end for**
8: Insert Q^* into Q_1
9: **return** (Q_1, Q_2)

Procedure 6.6 Confine($\mathbf{a}$, $\tau_{\mathbf{a}}$)

Input: $\mathbf{a}$, $\tau_{\mathbf{a}}$
Output: U^*

1: $workingQueue \leftarrow \mathbf{a}$;
2: **while** $workingQueue \neq \emptyset$ **do**
3: $\quad \mathbf{p} \leftarrow dequeue(workingQueue)$
4: $\quad e^* \leftarrow getAnEnabledEdge(\mathbf{p}, \tau_{\mathbf{a}})$
5: $\quad$ **if** $e^* = 0$ **then**
6: $\quad\quad$ Insert $\mathbf{p}$ into U^*
7: $\quad$ **else**
8: $\quad\quad$ **for all** $\mathbf{x} \in BlockEd[e^*]$ **do**
9: $\quad\quad\quad$ **if** Feasible($\lambda_{\{\mathbf{x},\mathbf{p}\}}$) **then**
10: $\quad\quad\quad\quad$ Insert $\lambda_{\{\mathbf{x},\mathbf{p}\}}$ into $workingQueue$
11: $\quad\quad\quad$ **end if**
12: $\quad\quad$ **end for**
13: $\quad$ **end if**
14: **end while**
15: **return** U^*

$\mathbf{u} \in U$ is inserted into Q_1 *iff* the set $Q_1 \cup Q_2$ does not contain any state dominated by $\mathbf{u}$. Furthermore, if $\mathbf{u}$ is dominated by a state $\mathbf{x} \in Q_1 \cup Q_2$, $\mathbf{x}$ is removed from $Q_1 \cup Q_2$.

Next, we present an outline of the $Confine$ operation introduced in Definition 6.4. The algorithmic steps of the $Confine$ operation are depicted in Procedure 6.6. The mechanism of the procedure is very similar to that of Lines 12-27 in Procedure 6.4, but instead of seeking to block enabled processing stages, the $Confine$ procedure seeks to block the enabled edges that do not belong to $\tau_{\mathbf{a}}$. The function *getAnEnabledEdge*

at Line 4 returns an enabled edge at state $\mathbf{p}$ that does not belong to $\tau_{\mathbf{a}}$. If no such edge exists, the function returns a value of 0, and thus, $\mathbf{p}$ is added to U^* at Line 6. On the other hand, if an enabled edge e^* is found, then the code of Lines 8-11 generates and processes every feasible combination of state $\mathbf{p}$ with the minimal states blocking e^*.

Now, we present the main content of the section, which is the algorithm used to enumerate the minimal reachable unsafe states. The complete logic of this algorithm is detailed in Algorithm 6.7. Algorithm 6.7 employs the queue Q to store unprocessed unsafe states, the list $\dot{U}$ to store processed unsafe states, and the hash table $\dot{A}$ to store boundary safe states. The algorithm starts by enumerating all the minimal reachable deadlock states using Procedure 6.4, and adds the returned states to Q. For each state $\mathbf{u}$ in Q, $\mathbf{u}$ is traced back by one transition in Line 6. Then, in Line 7, the states generated in Line 6 are partitioned into the sets $Safe\_Prev$ and $Unsafe\_Prev$ (i.e., the safe and unsafe state subsets of $Prev(\mathbf{u})$), using standard reachability analysis w.r.t. the target state $\mathbf{s}_0$. In Line 8, the elements of $Unsafe\_Prev$ are inserted into Q to be processed later. On the other hand, the function $Combine$ in Line 12 returns $\lambda_{\mathbf{a},\dot{\mathbf{a}}}$ if $\mathbf{a}$ and $\dot{\mathbf{a}}$ are combinable according to Definition 6.3. Otherwise, it returns $\emptyset$. Hence, in Lines 9-17, for each state $\mathbf{a} \in Safe\_Prev$, the $Combine$ function is applied with every state $\dot{\mathbf{a}} \in \dot{A}$, and the result is inserted in Z_a. In Line 14, Z_a is partitioned using standard reachability analysis into its subset of safe states, $Safe(Z_a)$, and its subset of unsafe states, $Unsafe(Z_a)$. As explained in Section 6.3.1, the states of $Safe(Z_a)$ are boundary safe states by construction; hence, they are inserted into $\dot{A}$ at Line 15. Whenever a boundary state $\mathbf{a'}$ is inserted into $\dot{A}$, we check first if $\exists \dot{\mathbf{a}} \in \dot{A}$ s.t. $\mathbf{a'} = \dot{\mathbf{a}}$; in this case $\tau_{\dot{\mathbf{a}}}$ is updated to $\tau_{\dot{\mathbf{a}}} \cup \tau_{\mathbf{a'}}$, and $\mathbf{a'}$ is discarded. On the other hand, the states of $Unsafe(Z_a)$ are unsafe; hence, they are inserted into Q. If Q is empty, then we apply the $Confine$ operation described in Procedure 6.6 to every state in $\dot{A}$ that has not been subjected to this operation yet.

Algorithm 6.7

Input: A RAS instance Φ

Output: the list $\dot{U}$ containing all the reachable minimal unsafe states of Φ

1: $\dot{U}, \dot{A} \leftarrow \emptyset$; $k \leftarrow 0$;
2: $Q \leftarrow EnumMinReachDeadlocks(\Phi)$
3: **while** $Q \neq \emptyset$ or $k < |\dot{A}|$ **do**
4: **if** $Q \neq \emptyset$ **then**
5: $\mathbf{u} \leftarrow$ dequeue(Q);
6: $Prev(\mathbf{u}) \leftarrow$ Backtrace($\mathbf{u}$);
7: $(Safe\_Prev,\ Unsafe\_Prev) \leftarrow$ Classify($Prev(\mathbf{u})$)
8: Insert_Non_Min($Unsafe\_Prev,\ Q,\ \dot{U}$)
9: **for** each $\mathbf{a} \in Safe\_Prev$ **do**
10: $Z_a \leftarrow \emptyset$
11: **for all** $\dot{\mathbf{a}} \in \dot{A}$ **do**
12: $Z_a \leftarrow Z_a \cup Combine(\dot{\mathbf{a}}, \mathbf{a})$
13: **end for**
14: $(Safe(Z_a), Unsafe(Z_a)) \leftarrow Classify(Z_a)$
15: Insert $\mathbf{a}$, $Safe(Z_a)$ into $\dot{A}$
16: Insert_Non_Min($Unsafe(Z_a),\ Q,\ \dot{U}$)
17: **end for**
18: Insert_Non_Min($\mathbf{u},\ \dot{U},\ Q$)
19: **else**
20: **while** $k < |\dot{A}|$ **do**
21: $\mathbf{a}_k \leftarrow \dot{A}[k++]$;
22: $U^* \leftarrow Confine(\mathbf{a}_k,\ \tau_{\mathbf{a}_k})$
23: Insert_Non_Min($U^*,\ Q,\ \dot{U}$)
24: **end while**
25: **end if**
26: **end while**
27: Remove_Unreachable($\dot{U}$)
28: **return** $\dot{U}$

The proposed algorithm terminates when there are no more unsafe states to be traced back nor any states to be confined. As illustrated by Procedure 6.5, whenever an element is inserted in $L = \dot{U} \cup Q$ via the function $Insert\_Non\_Min$, L is checked for dominance and it is updated accordingly. In particular, a state is deleted from L only if it dominates a newly generated state. Furthermore, such a state never enters L again. When combined with the finiteness of the underlying state space, the previous remarks establish that the presented algorithm terminates in a finite number of steps.

The (co-)reachability analysis that is performed in Lines 7, 14 and 27, is implemented using depth-first search supported with hash tables to mark visited states. In fact, we can skip this analysis in Lines 7 and 14. This variant of the proposed algorithm will assume that each state $\mathbf{x}$ that results from backtracing (Line 6) or *Combine* (Lines 10-13) is a safe state, unless $g(\mathbf{x}) = \tau_{\mathbf{x}}$, since in this case all the enabled events ($g(\mathbf{x})$) lead to unsafety (since they are members of $\tau_{\mathbf{x}}$). To describe the algorithm modifications in this alternative approach, let $\mathbf{x}_1$ be a state that results from backtracing $\mathbf{u}$, and let $\mathbf{x_2}$ be a state that results from the *Combine* operation. If $g(\mathbf{x_1}) = \tau_{\mathbf{x_1}}$, then $\mathbf{x}_1$ is added to $Unsafe\_Prev$. Otherwise, it is added to $Safe\_Prev$, i.e., it is assumed to be safe. Similarly, if $g(\mathbf{x_2}) = \tau_{\mathbf{x_2}}$, then $\mathbf{x}_2$ is added to $Unsafe(Z_a)$. Otherwise, it is added to $Safe(Z_a)$. It was shown in [56] that this alternative approach enumerates correctly all the minimal reachable unsafe states.

We tried both approaches, and it turned out that the first one – i.e., the one presented in the main statement of Algorithm 6.7 – is much more computationally efficient. This can be explained by the following two reasons:

1. As revealed in the sequel, the complexity of the algorithm is highly dependent on $|\dot{A}|$. In the second approach, the list $\dot{A}$ grows much larger because it contains both safe and unsafe states and their combinations, whereas in the first approach, $\dot{A}$ contains only safe states.

2. The states that are evaluated for co-reachability, are either minimal unsafe states or few steps away from minimal unsafe states. Thus, the examined states are characterized by a low process content, and the applied depth-first search method is computationally very efficient.

The correctness of Algorithm 6.7 is formally proven in the last part of the section. Next, we present two examples that demonstrate the respective application of the *Combine* and the $Confine$ operations.

Example 6.2 Consider the RAS configuration depicted in Table 6.5. It has four resource types, $\{R_1, \ldots, R_4\}$, all with a single unit capacity. It also has three process types, $\{J_1, J_2, J_3\}$, all with simple linear structure and single-type resource allocation. The Finite State Automaton (FSA) representing the reachable state space corresponding to the considered RAS is depicted in Figure 6.2. As it is revealed by the figure, the RAS has three minimal reachable deadlock states $\mathbf{u}_1 = \{\Xi_{12}, \Xi_{21}\}$, $\mathbf{u}_2 = \{\Xi_{22}, \Xi_{31}\}$ and $\mathbf{u}_3 = \{\Xi_{11}, \Xi_{32}\}$, and one minimal reachable unsafe state $\mathbf{u}_4 = \{\Xi_{11}, \Xi_{21}, \Xi_{31}\}$. The next states of $\mathbf{u}_4$ are $\mathbf{nu}_1 = \{\Xi_{12}, \Xi_{21}, \Xi_{31}\}$, $\mathbf{nu}_2 = \{\Xi_{11}, \Xi_{22}, \Xi_{31}\}$, and $\mathbf{nu}_3 = \{\Xi_{11}, \Xi_{21}, \Xi_{32}\}$. It can be seen that $\mathbf{nu}_1 \succ \mathbf{u}_1$, $\mathbf{nu}_2 \succ \mathbf{u}_2$, and $\mathbf{nu}_3 \succ \mathbf{u}_3$. Applying Procedure 6.4 results in the minimal reachable deadlock states $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$. The main idea underlying Algorithm 6.7 is to backtrace from the three minimal reachable deadlock states and to combine the resultant boundary safe states in order to generate the minimal reachable unsafe state $\mathbf{u}_4$. Table 6.6 depicts the minimal reachable unsafe states obtained by applying Algorithm 6.7, and the result of backtracing from each of them. The boldfaced processing stages indicate the source nodes of the backtraced edges in $\tau_{\mathbf{a}}$. As a more concrete example, consider state $\mathbf{u}_1 = \{\Xi_{11}, \Xi_{32}\}$. Ξ_{11} cannot be traced back because this is an initiating stage. Backtracing on $(\Xi_{31} \rightarrow \Xi_{32})$ yields state $\mathbf{a}_1 = \{\Xi_{11}, \mathbf{\Xi_{31}}\}$, which is a safe state. The algorithm starts by inserting $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$ into Q. First, $\mathbf{u}_1$ is backtraced, adding state $\mathbf{a}_1$ to $\dot{A}$. Next, $\mathbf{u}_2$ is backtraced, generating the boundary safe state $\mathbf{a}_2$. Assessing the combinability of $\mathbf{a}_1$ and $\mathbf{a}_2$, we can see that $\lambda_{\{\mathbf{a_1}, \mathbf{a_2}\}} = \{\Xi_{11}, \Xi_{31}, \Xi_{21}\}$, with the enabled edges $g(\lambda_{\{\mathbf{a}_1, \mathbf{a}_2\}}) = \{\Xi_{11} \rightarrow \Xi_{12}, \Xi_{21} \rightarrow \Xi_{22}, \Xi_{31} \rightarrow \Xi_{32}\}$ and $\tau_{\lambda_{\{\mathbf{a}_1, \mathbf{a}_2\}}} = \{\Xi_{11} \rightarrow \Xi_{12}, \Xi_{31} \rightarrow \Xi_{32}\}$. Hence, $\mathbf{a_1}$ and $\mathbf{a_2}$ are combinable. Moreover, $\lambda_{\{\mathbf{a_1}, \mathbf{a_2}\}}$ is an unsafe state. Hence, it is added to $Unsafe(Z_a)$, and consequently into Q. The same process is repeated with $\mathbf{u}_3$, but combining $\mathbf{a}_3$ with each of $\mathbf{a}_1$ and $\mathbf{a}_2$ yields $\mathbf{u}_4$ again. Hence, $\mathbf{u_4}$ can be constructed by applying the *Combine* operation to any pair of $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$. On the other hand, $\mathbf{u}_4$ can not be traced back because all its processing stages are initiating stages. Thus,

Table 6.5: The RAS considered in Example 6.2

Resource Types:	$\{R_1, \ldots, R_4\}$
Resource Capacities:	$C_i = 1, \forall i \in \{1, \ldots, 4\}$
Process Type 1:	$R_1 \rightarrow R_2 \rightarrow R_3$
Process Type 2:	$R_3 \rightarrow R_2 \rightarrow R_4$
Process Type 3:	$R_4 \rightarrow R_2 \rightarrow R_1$

Table 6.6: Tracing back from the minimal reachable unsafe states for Example 6.2

$\mathbf{u}$	$Safe\_Prev$	$Unsafe\_Prev$
$\mathbf{u}_1 = \{\Xi_{11}, \Xi_{32}\}$	$\mathbf{a}_1 = \{\Xi_{11}, \boldsymbol{\Xi_{31}}\}$	$\emptyset$
$\mathbf{u}_2 = \{\Xi_{12}, \Xi_{21}\}$	$\mathbf{a}_2 = \{\boldsymbol{\Xi_{11}}, \Xi_{21}\}$	$\emptyset$
$\mathbf{u}_3 = \{\Xi_{22}, \Xi_{31}\}$	$\mathbf{a}_3 = \{\boldsymbol{\Xi_{21}}, \Xi_{31}\}$	$\emptyset$
$\mathbf{u}_4 = \{\Xi_{11}, \Xi_{21}, \Xi_{31}\}$	$\emptyset$	$\emptyset$

after the fourth iteration, $Q = \emptyset$ and $\dot{A} = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$. Applying the $Confine$ operation to the elements of $\dot{A}$ does not generate any minimal unsafe state. Hence, the algorithm terminates.

Before we conclude the example, we also consider the alternative approach that skips the co-reachability analysis of the generated states. After the generation of $\mathbf{u}_4$ by $Combine(\mathbf{a}_2, \mathbf{a}_1)$, $\mathbf{u}_4$ is inserted into $\dot{A}$ because $g(\lambda_{\{\mathbf{a}_1, \mathbf{a}_2\}}) \not\subseteq \tau_{\lambda_{\{\mathbf{a}_1, \mathbf{a}_2\}}}$. Also, $\tau_{\mathbf{u}_4} = \{\Xi_{11} \rightarrow \Xi_{12}, \Xi_{31} \rightarrow \Xi_{32}\}$. In the next iteration, $\mathbf{u}_4$ is generated three more times as a result of $Combine(\mathbf{a}_3, \mathbf{a}_1)$, $Combine(\mathbf{a}_3, \mathbf{a}_2)$ and $Combine(\mathbf{a}_3, \mathbf{u}_4)$. But only at $\mathbf{u}_4 = \lambda_{\{\mathbf{a}_3, \mathbf{u}_4\}}$ we get $g(\lambda_{\{\mathbf{a}_3, \mathbf{u}_4\}}) = \tau_{\lambda_{\{\mathbf{a}_3, \mathbf{u}_4\}}}$; at that point, $\mathbf{u}_4$ is eventually classified as an unsafe state. Thus, it took the alternative approach one more iteration, one more element in $\dot{A}$, and three more $Combine$ operations to realize that $\mathbf{u}_4$ is unsafe. □

Example 6.3 To illustrate the application of the $Confine$ operation, consider the RAS configuration depicted in Table 6.7. The considered RAS has four resource types, $\{R_1, \ldots, R_4\}$, all with a single unit capacity. It also has three process types, $\{J_1, J_2, J_3\}$, all with simple linear structure and single-type resource allocation, except

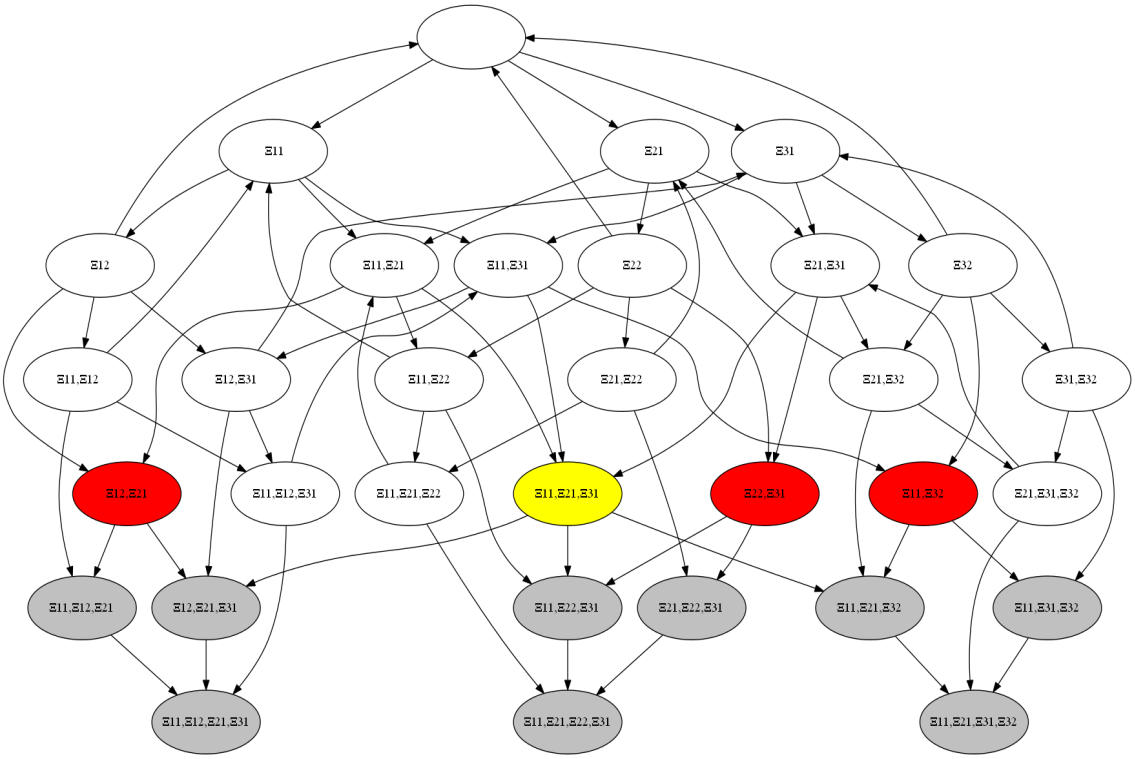


Figure 6.2: The FSA representing the reachable state space of the RAS depicted in Table 6.5. The white states represent the safe states. The red states represent the minimal reachable deadlock states. The yellow state represents the minimal reachable deadlock-free unsafe state. The gray states represent the non-minimal reachable unsafe states.

Table 6.7: The RAS considered in Example 6.3

Resource Types:	$\{R_1, \ldots, R_4\}$
Resource Capacities:	$C_i = 1, \forall i \in \{1, \ldots, 4\}$
Process Type 1:	$R_1 \rightarrow R_2 \rightarrow R_3$
Process Type 2:	$R_3 \rightarrow \{R_2,\ R_4\}$
Process Type 3:	$R_4 \rightarrow R_1$

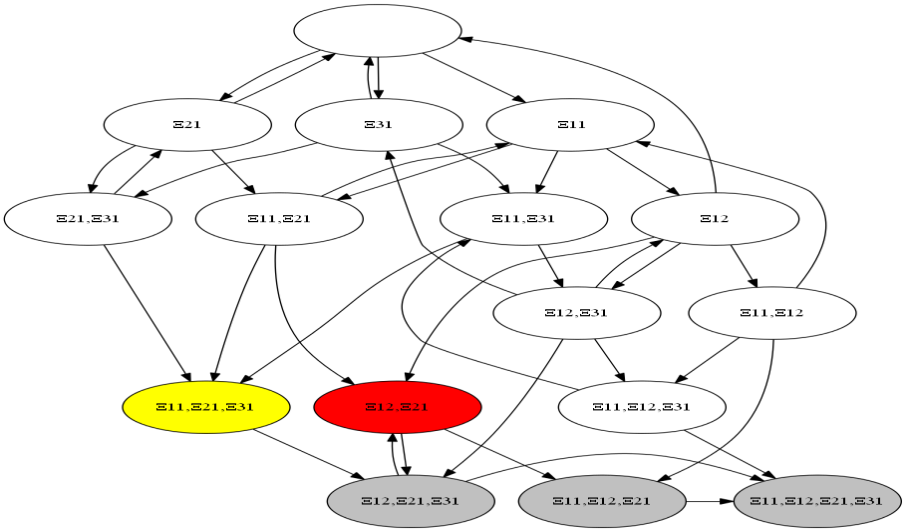


Figure 6.3: The FSA representing the reachable state space of the RAS depicted in Table 6.7. The white states represent the safe states. The red states represent the minimal reachable deadlock states. The yellow state represents the minimal reachable deadlock-free unsafe state. The gray states represent the non-minimal reachable unsafe states.

for processing stage Ξ_{21} which requests the allocation of, both, R_2 and R_4. The FSA representing the reachable state space of the considered RAS is depicted in Figure 6.3. The application of Procedure 6.4 yields the minimal reachable deadlock state $\mathbf{u}_1 = \{\Xi_{12}, \Xi_{21}\}$. Backtracing from $\mathbf{u}_1$ yields $\mathbf{a}_1 = \{\Xi_{11}, \Xi_{21}\}$, which is a safe state: $\tau_{\mathbf{a}_1} = (\Xi_{11} \rightarrow \Xi_{12})$, whereas $g(\mathbf{a}_1) = \{\Xi_{11} \rightarrow \Xi_{12}, \Xi_{21} \rightarrow \Xi_{22}\}$. Applying the $Confine$ operation to $\mathbf{a}_1$ to block the edge $(\Xi_{21} \rightarrow \Xi_{22})$ yields the second minimal reachable unsafe state $\mathbf{u}_2 = \{\Xi_{11}, \Xi_{21}, \Xi_{31}\}$. □

Complexity analysis: We start by analyzing the following simple operations: (i) As mentioned before, checking the feasibility of a given state has complexity $\mathcal{O}(\mu \cdot \xi)$.

(ii) Determining the set of feasible events in a given state has complexity $\mathcal{O}(|\mathcal{E}| \cdot \mu \cdot \xi)$. (iii) Based on the previous item, assessing the combinability (c.f. Def. 6.3) of a pair of states can be supported with $\mathcal{O}(|\mathcal{E}| \cdot \mu \cdot \xi)$. (iv) Backtracing a state has complexity $\mathcal{O}(|\mathcal{E}|)$. (v) Based on the first and the fourth item, generating the previous states of a given state has complexity $\mathcal{O}(|\mathcal{E}| \cdot \mu \cdot \xi)$.

Let α denote the total number of unsafe states added to Q throughout the algorithm. The total number of boundary safe states generated by backtracing unsafe states is upper bounded by $\mathcal{O}(\alpha \cdot |\mathcal{E}|)$. Since $\dot{A}$ is a hash table, inserting an element into $\dot{A}$ in the way described in the algorithm is of order $\mathcal{O}(1)$. Moreover, since $\dot{A}$ is a subset of the power set of the boundary safe states generated throughout the algorithm, the size of the list $\dot{A}$ is bounded above by $\mathcal{O}(2^{\alpha \cdot |\mathcal{E}|})$. Combining this bound with the third item in the previous paragraph, we can see that the total running time of Line 12 is $\mathcal{O}(|\dot{A}| \cdot \mu \cdot \xi \cdot |\mathcal{E}|) = \mathcal{O}(2^{\alpha \cdot |\mathcal{E}|} \cdot \mu \cdot \xi \cdot |\mathcal{E}|)$.

Analyzing the $Confine$ function depicted in Procedure 6.6, we can see that the number of the state combinations that are constructed by the procedure is $\mathcal{O}(\prod_{e=1}^{|\mathcal{E}|} |BlockEd[e]|) = \mathcal{O}(\rho^{|\mathcal{E}|})$. Getting an enabled edge at a given state has complexity $\mathcal{O}(|\mathcal{E}| \cdot \mu \cdot \xi)$. Hence, the complexity of the $Confine$ procedure is $\mathcal{O}(\rho^{|\mathcal{E}|} \cdot \mu \cdot \xi \cdot |\mathcal{E}|)$. Therefore, the total running time of Line 22 of Algorithm 6.7 is $\mathcal{O}(\rho^{|\mathcal{E}|} \cdot \mu \cdot \xi \cdot |\mathcal{E}| \cdot |\dot{A}|)$.

The reachability analysis performed at Lines 7, 14 and 27 has complexity $\mathcal{O}(|S_r|)$. Finally, the total running time of the pairwise vector comparisons that are performed by the function Insert_Non_Minimal, is $\mathcal{O}(\alpha^2 \cdot \xi)$.

We remind the reader that the complexity of the function $EnumMinReachDeadlocks$ (Procedure 6.4) is $\mathcal{O}(\rho^{2 \cdot |\mathcal{E}|} \cdot \xi + |S_r|)$. Summing up all the results that are obtained from the above analysis, we can conclude that the complexity of Algorithm 6.7 is $\mathcal{O}(\rho^{2 \cdot |\mathcal{E}|} \cdot \xi + |S_r|)$ (Line 2)+$\mathcal{O}(\alpha \cdot |\mathcal{E}| \cdot \mu \cdot \xi)$ (Line 6) +$\mathcal{O}(|S_r|)$ (Lines 7, 14, 27) + $\mathcal{O}(2^{\alpha \cdot |\mathcal{E}|} \cdot \mu \cdot \xi \cdot |\mathcal{E}|)$ (Line 12) + $\mathcal{O}(\rho^{|\mathcal{E}|} \cdot \mu \cdot \xi \cdot |\mathcal{E}| \cdot 2^{\alpha \cdot |\mathcal{E}|})$ (Line 22) + $\mathcal{O}(\alpha^2 \cdot \xi)$ (Lines

8, 16, 18, 23) $\approx \mathcal{O}(\rho^{|\mathcal{E}|} \cdot \mu \cdot \xi \cdot |\mathcal{E}| \cdot 2^{\alpha \cdot |\mathcal{E}|} + |S_r|) = \mathcal{O}(\mu^{|\mathcal{E}|+1} \cdot \eta^{C \cdot |\mathcal{E}|} \cdot \xi \cdot |\mathcal{E}| \cdot 2^{\alpha \cdot |\mathcal{E}|} + |S_r|)$. Hence, the computational complexity of the algorithm is particularly sensitive to the capacities of the resource types, the number of the distinct event types, and the number of enumerated unsafe states. On the other hand, as we shall see from the computational experiments presented in Section 6.4, statistically, the algorithm running time is mostly correlated with the size of the list $\dot{A}$.

6.3.3 Proving the correctness of Algorithm 6.7

The finite termination of Algorithm 6.7 has already been established in the previous section. Hence, to establish the algorithm correctness, it remains to show that it generates all the minimal reachable unsafe states. Theorem 6.1 proves that Procedure 6.4 enumerates all the minimal reachable deadlock states. Thus, our goal is to show that Algorithm 6.7 enumerates also all the deadlock-free minimal reachable unsafe states. The main idea is to show that a minimal unsafe state is missed only if a "next" minimal unsafe state – i.e., a member of the set $nextMin$ introduced in Definition 6.2 – is also missed by the algorithm. Then, the algorithm correctness can be based upon the elimination of the possibility of having cyclic paths among minimal unsafe states.

Proposition 6.2 *A deadlock-free minimal unsafe state* $\mathbf{u}$ *is missed by Algorithm 6.7 only if an entire set of states* $nextMin(\mathbf{u}, e_i)$ *is missed by the algorithm.*

Proof: Suppose that the algorithm generates at least one element from each set $nextMin(\mathbf{u}, e_i)$, and w.l.o.g further assume that $\{\mathbf{h}_1, \ldots, \mathbf{h}_k\}$ are minimal unsafe states and $\{\mathbf{h}_{k+1}, \ldots, \mathbf{h}_K\}$ are non-minimal unsafe states, $0 \leq k \leq K$. By the working assumption, we can consider that the algorithm generates the states $\{\mathbf{h}_1, \ldots, \mathbf{h}_k, \mathbf{z}_{k+1,1}, \ldots, \mathbf{z}_{K,1}\}$, where the notation of Definition 6.2 has been applied. The set $Y = \{\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(k)}, \mathbf{s}_{k+1,1}, \ldots, \mathbf{s}_{K,1}\}$ collects the corresponding predecessors of the states in the set $\{\mathbf{h}_1, \ldots, \mathbf{h}_k, \mathbf{z}_{k+1,1}, \ldots, \mathbf{z}_{K,1}\}$, that are obtained by tracing back from each of the elements of the set $\{\mathbf{h}_1, \ldots, \mathbf{h}_k, \mathbf{z}_{k+1,1}, \ldots, \mathbf{z}_{K,1}\}$ respectively across the edges

$e_1, \ldots, e_K$. It can be seen that $\mathbf{u}^{(1)} = \ldots = \mathbf{u}^{(k)} = \mathbf{u}$, $\tau_{\mathbf{u}^{(i)}} = \{e_i\},\ i \leq k$, and $\tau_{\mathbf{s}_{i1}} = \{e_i\},\ i > k$. If $k \geq 1$, then $\mathbf{u}$ is the predecessor of $\mathbf{h}_1$ that is obtained by tracing back e_1; hence, $\mathbf{u}$ is generated by Lines 6-7 of Algorithm 6.7, and we are done. On the other hand, if $k = 0$, then $\forall i = 1 : K,\ \mathbf{s}_{i1}$ is a boundary safe state obtained by tracing back e_i from the unsafe state $\mathbf{z}_{i1}$. Definition 6.2 and its accompanying discussion reveal that $\mathbf{s}_{i1} \prec \mathbf{u},\ \forall i$. Hence, $\mathbf{a}_2 = \lambda_{\{\mathbf{s}_{11}, \mathbf{s}_{21}\}} \preceq \mathbf{u}$. Therefore, we can infer that e_1 and e_2 are enabled at $\mathbf{a}_2$. Hence, $\mathbf{s}_{11}$ and $\mathbf{s}_{21}$ are combinable, and $\tau_{\mathbf{a_2}} = (\tau_{\mathbf{s}_{i1}} \cup \tau_{\mathbf{s}_{i2}}) \cap g(\mathbf{a}_2) = \{e_1, e_2\}$. If $\mathbf{a}_2$ is unsafe, then, by the minimality of $\mathbf{u}$, $\mathbf{a}_2 = \mathbf{u}$. Hence, $\mathbf{u}$ is added to $Unsafe(Z_a)$ at Line 14, and consequently to Q at Line 16. Otherwise, $\mathbf{a}_2$ is added to $Safe(Z_a)$ at Line 14, and consequently to $\dot{A}$ at Line 15. In a subsequent iteration, Line 12 will find that $\mathbf{s}_{31}$ and $\mathbf{a}_2$ are also combinable, yielding $\mathbf{a}_3 = \lambda_{\{\mathbf{s}_{31}, \mathbf{a}_2\}}$. The same argument is repeated until either $\mathbf{u}$ or $\mathbf{a}_K \equiv \lambda_{\{\mathbf{s}_{11}, \ldots, \mathbf{s}_{K1}\}} \prec \mathbf{u}$ is generated. It also holds that $\tau_{\mathbf{a}_K} = \{e_1, \ldots, e_K\}$. Hence, we can see that, if $\mathbf{u}$ has not been generated during the $K-1$ "combine" iterations that generate $\mathbf{a}_K$, by the definition of the $Confine$ operation, it will be contained in the set of states returned by $Confine(\mathbf{a}_K, \tau_{\mathbf{a}_K})$, and it is thereby generated by Line 22. □

The next lemma eliminates the possibility of having cyclic paths among minimal unsafe states.

Lemma 6.2 *Consider a sequence of minimal unsafe states* $\{\mathbf{u}_k\}_{k=1}^{l}$ *such that* $\mathbf{u}_k \in nextMin(\mathbf{u}_{k-1}),\ k = 2 : l$. *Then* $\mathbf{u}_1 \notin nextMin(\mathbf{u}_l)$.

Proof: Let $e_1 = (\Xi_{ij}, \Xi_{ij^*})$ be the transition edge between $\mathbf{u}_1$ and its next state in the considered sequence. Let $\vec{\Xi}_{ij}$ denote the processing stages in $\mathcal{G}_i$ that are co-reachable to Ξ_{ij}; by definition, $\Xi_{ij} \in \vec{\Xi}_{ij}$. The acyclicity of $\mathcal{G}_i$ (c.f. Assumption 6.1) implies that $\sum_{\Xi_{ij'} \in \vec{\Xi}_{ij}} \mathbf{u}_2(\Xi_{ij'}) \leq \sum_{\Xi_{ij'} \in \vec{\Xi}_{ij}} \mathbf{u}_1(\Xi_{ij'}) - 1$. Due to the absence of loading events from the considered sequence and the acyclicity of $\mathcal{G}_i$, there does not exist a state

$\mathbf{u}_k$, $k \geq 2$, in this sequence where $\sum_{\Xi_{ij'} \in \vec{\Xi}_{ij}} \mathbf{u}_k(\Xi_{ij'})$ is restored to its original level at $\mathbf{u}_1$. Therefore, $\mathbf{u}_1 \notin nextMin(\mathbf{u}_l)$. $\square$

Now we are ready to present the main result regarding the correctness of Algorithm 6.7.

Theorem 6.2 *Algorithm 6.7 generates all the minimal reachable unsafe states.*

Proof: Assume that the minimal reachable unsafe state $\mathbf{u}_1$ is missed by the algorithm. Proposition 6.2 ensures that this is due to missing another minimal unsafe state $\mathbf{u}_2 \in nextMin(\mathbf{u}_1)$. Repeating the same argument with $\mathbf{u}_2$, we get that the algorithm missed the minimal unsafe states $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_l$, where $\mathbf{u}_k \in nextMin(\mathbf{u}_{k-1})$. But Lemma 6.2 eliminates the possibility of having a cycle in this path and l is bounded from above by $|\hat{S}_{\bar{s}}|$. Hence, $\mathbf{u}_l$ is a missed minimal deadlock state. $\mathbf{u}_l$ is also reachable, by the specification of the sequence $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_l$, and the fact that, in the considered RAS class, $\mathbf{s} \in S_r \wedge \mathbf{s}' \preceq \mathbf{s} \Longrightarrow \mathbf{s}' \in S_r$. But when combined with Lines 2 and 18 of Algorithm 6.7, the above results contradict Theorem 6.1. $\square$

6.4 Computational results

In this section we report the results from a series of computational experiments, in which we applied Algorithm 6.7 upon a number of randomly generated instantiations of the RAS class considered in this chapter. Moreover, we compare the performance of this algorithm with both the standard search-type enumeration algorithm and the algorithm of Appendix B. The latter algorithms rely on the exhaustive enumeration of the underlying state space. Each of the generated instances was further specified by:

- The number of resource types in the system; the range of this parameter was between 3 and 16.
- The capacities of the resource types in the system; the range of this parameter

was between 1 to 4. Moreover, for a given instance, all the resource types have the same capacity.

- The number of process types in the system; the range of this parameter was between 3 and 5.

- The structure of the process graphs $\mathcal{G}_i$ and of the resource request vectors that were supported by the resource allocation function $\mathcal{A}$. In terms of the first attribute, the generated instances contains RAS where all their processes presented a simple linear structure, and RAS where some of their processes possessed routing flexibility (Disjunctive RAS). In terms of the second attribute mentioned above, the generated instances contain RAS with single-type resource allocation and instances with conjunctive resource allocation (Conjunctive RAS).

- The number of processing stages in each process; the range of this parameter was between 3 and 16. Furthermore, in order to remain consistent with the RAS structure defined in Section 1.1, no processing stage has a zero resource-allocation vector.

The employed RAS generator was encoded and compiled in C++. For each generated RAS instance, Φ, we enumerated the minimal reachable unsafe states by applying: (i) the standard search-type enumeration algorithm, (ii) the algorithm of Appendix B, and (iii) Algorithm 6.7. We imposed a hard limit of 48 hours for the solution of these instances. All our computational experiments were performed on a 2.66 GHz quad-core Intel Xeon 5430 processor with 6 MB of cache memory and 32 GB RAM; however, each job ran on a single core. The aforementioned three algorithms were all encoded in C++, compiled and linked by the GNU g++ compiler under Unix.

Table 6.8: A sample of our computational results regarding the efficacy of Algorithm 6.7

C	ξ	μ	$\lvert S_r \rvert$	$S_{r\bar{s}}$	$\hat{S}_{r\bar{s}}$	γ	α	$\dot{A}$	t_s	t_a	t_n	t_r
1	36	10	635572	349546	1654	2807	5912	8548	1225	224	69	0
1	30	14	1463878	458127	284	540	868	2124	1418	643	6	0
1	42	8	404542	237591	3475	6419	12848	9210	2002	175	73	0
1	32	13	1508301	573055	686	1076	3017	5792	2633	101	35	0
1	39	9	799071	401974	7568	11753	20736	15534	4759	898	199	0
1	39	11	1743534	887064	2840	4130	18516	33020	13270	784	555	0
1	45	9	1659342	987461	10468	21410	45139	39937	16690	2490	1447	1
1	42	9	1962454	1098441	7171	11002	24859	42639	21001	2136	989	0
1	40	13	4488904	2748034	658	2088	8384	23095	29765	723	259	0
1	48	10	3436211	1789990	36874	68317	135762	91164	105074	27593	5863	1
1	42	12	14158338	4977105	35022	54327	116337	136038	229759	136040	31933	10
1	45	12	74649601	4007464	30110	59864	169071	234629		170871	36305	4
1	45	13			33744	72935	193572	274553			44222	5
1	45	14			22157	43718	129210	190378			23382	7
1	51	11			20891	39738	165495	299266			51295	4
1	44	14	51954392	17158379	16910	29819	82621	138687		171919	14851	3
3	21	8	646746	175449	779	849	1815	908	167	76	2	0
3	21	11	1767552	406300	559	660	1050	4729	541	461	20	0
1	30	8	915716	412871	849	1103	6065	23837	1551	176	509	0
1	27	6	738720	644955	4162	4614	10752	8646	2251	103	327	0
3	28	9	2939463	1328411	3655	5928	10619	11354	5060	1345	193	0
2	27	8	2430581	867647	14185	16547	29722	38765	6774	2559	1112	0
2	42	10	1962454	1098441	7171	11002	24859	42639	12087	2059	1073	0
2	30	7	1712672	977484	7214	10617	32861	54891	13027	873	1782	0
3	40	9	3554952	2366757	2533	3659	17855	19856	21082	1305	614	0
3	33	9	6051299	2300151	5069	6837	24373	65602	34650	2174	1865	0
4	35	10	24430444	7268845	9783	11307	32410	90665	114614		5339	1
3	42	11	13335839	5237457	3145	6998	20843	27747		1041	550	0
1	42	14	110100471	34796004	16910	29819	82621	138687		161475	12649	3
4	45	13			33744	72935	193572	274553			45550	5
4	32	10			4046	4393	6592	7585			92	0
3	33	14			428	585	2957	31425			1098	0
4	18	12	571536	0	0	0	0	0	19	420	0	0
3	24	12	2171880	13992	28	28	32	126	48	253	0	0
5	30	11	1229688	64968	241	271	279	373	105	354	0	0
4	27	10	2693250	76536	79	79	87	37	130	476	0	0
4	28	14	3416000	280000	1	1	8	1	158	445	0	0
3	24	14	2448000	410112	9	9	9	50	580	408	0	0
1	42	10	1663534	230889	2665	5857	6966	5030	801	672	173	0
1	44	11	2340408	511277	1522	2620	3376	3760	4455	1543	37	0
2	33	9	7885856	605977	2323	2628	2710	4882	4871	2434	19	0
1	42	14			17394	24192	27412	25199			1001	4
4	45	9			381	516	691	2729			77	0
4	24	13			1215	1245	1294	3490			15	0
3	35	13			24	31	31	77			747	0
4	36	12			6635	8543	13382	42510			665	2
4	36	14			792	1975	2000	2920			6	4
4	39	12			2468	6035	7444	12777			88	19

Table 6.8 reports a representative sample of the results that we obtained in our experiments. The first section of the table is for RAS instances with simple linear structure, single-type resource allocation, and single-unit resource types. The second section is for RAS instances with simple linear structure, conjunctive resource allocation, and multiple-unit resource types. Finally, the last section of the table is for RAS

instances with routing flexibility, conjunctive resource allocation, and multiple-unit resource types. Columns 1–3 in Table 6.8 report, respectively, the capacity of the resource types, the total number of processing stages, and the number of resource types in the corresponding RAS instance. On the other hand, Columns 4 – 6 report, respectively, the cardinalities of the set of reachable states, the set of the reachable unsafe states, and the set of minimal reachable unsafe states. Column 7 reports (γ) the number of minimal reachable unsafe states generated by Algorithm 6.7 without performing the reachability evaluation of the generated states (Line 27). Column 8 reports (α) the total number of unsafe states added to Q throughout the course of the execution of Algorithm 6.7. Column 9 reports the cardinality of the list $\dot{A}$ at the end of the execution of Algorithm 6.7. Column 10 reports (t_s) the amount of computing time (in seconds) that was required to compute the minimal reachable unsafe states through the standard search-type enumeration algorithm. On the other hand, Column 11 reports (t_a) the amount of computing time (in seconds) that was required to compute the minimal reachable unsafe states through the algorithm of Appendix B. Finally, Columns 12 (t_n) and 13 (t_r) report, respectively, the amount of time (in seconds) spanned by Lines 1-26 and Line 27 of Algorithm 6.7. The rows that have some unreported entries correspond to RAS instances for which the corresponding algorithm did not conclude within 48 hours. The reported cases are ordered in increasing magnitude of the corresponding t_s.

As mentioned in the previous paragraph, the data provided in Table 6.8 are representative of a more extensive sample that was collected in our experiments. The perusal of these data reveals very clearly

- the computational efficacy of Algorithm 6.7 in computing the minimal reachable unsafe states when compared to the standard search-type enumeration algorithm, and, for most of the cases, to the algorithm of Appendix B;

- the fact that the computational complexity of Algorithm 6.7 is mostly dependent

on the cardinality of the set $\dot{A}$;

- the fact that Algorithm 6.7 demonstrates more computational efficacy compared to the standard search-type enumeration algorithm and to the algorithm of Appendix B for RAS configurations with routing flexibility (this can be explained by the relative scarcity of unsafe states in RAS with routing flexibility);

- the fact that in spite of the high theoretical worst-case complexity of the computation in Line 26 of Algorithm 6.7 (assessing state reachability), its practical complexity is very benign.

6.5 Concluding remarks

This chapter has presented a novel algorithm for an efficient enumeration of the set of minimal reachable unsafe states that avoids the complete enumeration of the corresponding state space. Furthermore, through a series of experiments, we have demonstrated the computational efficiency of the proposed algorithm when compared to the traditional enumeration algorithms that rely upon the complete enumeration of the underlying state space.

CHAPTER VII

MAXIMALLY PERMISSIVE DEADLOCK AVOIDANCE FOR RESOURCE ALLOCATION SYSTEMS WITH R/W LOCKS

7.1 Introduction

As it was mentioned in Chapter 1, the supervisory control theory has been employed and extended in order to address deadlock-related issues that arise in the context of multi-threaded programs, a programming paradigm that becomes more and more prevalent due to the multi-core architectures that are promoted by the computer industry. The methodologies proposed in the previous chapters can be employed to address the problem of deadlock avoidance in multi-threaded programs where the concurrently running threads coordinate their execution in their critical section through mutual exclusion locks (mutexes) and semaphores. In this chapter, we further extend the aforementioned results in order to address the problem of deadlock avoidance for multi-threaded programs involving reader/writer (R/W-) locks, besides mutexes and semaphores. Reader-writer synchronization, as introduced in Chapter 1, relaxes the constraint of mutual exclusion to permit more than one process to inspect a shared resource concurrently, as long as none of them changes its value. Thus, multiple threads can read from the shared resource simultaneously but a thread can write to the shared resource only if no other thread is writing to, or reading from, this resource. We shall characterize this effect by saying that, when perceived as resources, R/W-locks work in two modes: (i) a "writing" mode where the resource capacity is equal to one, and (ii) a "reading" mode where the capacity is infinite. We shall refer to a RAS that contains R/W resource types as a R/W-RAS, whereas by "*conventional* " RAS, we

shall refer to the RAS class defined in Chapter 1.

The novel attributes exhibited by the R/W-locks have profound implications for the behavior of the R/W-RAS, differentiating them significantly from the previously studied RAS, and complicating their analysis and their management. More specifically, due to the infinite capacity of the R/W-lock when operating in the reading mode, it might not be possible to model R/W-RAS with Finite State Automata or bounded Petri nets. As a result, their behavioral analysis is not immediately amenable to the elementary, enumerative techniques that can provide the basic characterization for deadlock and deadlock avoidance in the case of the conventional RAS. In fact, the logical behavior of R/W-RAS cannot be modeled even by unbounded Petri nets (when staying within the basic definition of this formalism). Indeed, a process seeking the allocation of R/W-locks must consider not only the availability of their capacity, but also their current operational mode; in particular, a writing stage can be performed only if there are no active readers allocated the corresponding lock. In view of the infinite capacity of R/W-locks in their reading mode, the latter test can be supported only through the introduction of inhibitor arcs in the underlying PN model [67]. Thus, the past PN-based structural approaches for the synthesis of a deadlock avoidance policy (e.g. [22, 35]) are not transferrable to the new RAS model.

Yet, in the rest of this chapter, we establish that the maximally permissive DAP is effectively computable for R/W-RAS, in spite of all the aforementioned challenges. The key enabler of this result is the monotonicity property of the unsafe states introduced in Section 2.5. Due to that property, the set of unsafe states will admit an effective representation as long as its minimal elements are effectively enumerable. Hence, the key results of this chapter comprise: (i) the introduction of an automaton-based modeling framework for R/W-RAS, and (ii) and an effective algorithm for enumerating all the minimal reachable unsafe states in this representational framework. Once the set of minimal states is available, maximally permissive liveness-enforcing

supervision can be effected through techniques similar to those discussed in Chapter 5 for the conventional RAS. More specifically, a non-parametric classifier in the form of n-ary decision diagrams can be constructed to efficiently store the minimal reachable unsafe states generated by the introduced algorithm, in a way that facilitates the online assessment of the state-safety property.

In the light of the above discussion, the rest of the chapter is organized as follows: Section 7.2 provides a formal characterization of the R/W-RAS class considered in this chapter. Section 7.3 shows that the set of minimal reachable unsafe states is finite, and outlines the conditions and some possible methods for its enumeration. Section 7.4 details the particular algorithm that we propose for enumerating the minimal reachable unsafe states of R/W-RAS. Section 7.5 demonstrates the application of the proposed algorithm using an example. Finally, Section 7.6 concludes the chapter.

7.2 The considered R/W-RAS class

In this section, we extend the conventional RAS class defined in Chapter 1 to encompass the R/W locks and its acquisition dynamics. An instance Φ_{rw} from the R/W-RAS class is defined as a 5-tuple $\langle \mathcal{R}, \mathcal{RW}, C, \mathcal{P}, \mathcal{A} \rangle$. Basically, the characteristics and the dynamics of the R/W-RAS are identical to those of the conventional RAS except for the additional dynamics of the R/W locks. Hence, $\mathcal{R}$, C, and $\mathcal{P}$ are identical to their counterparts in the conventional RAS definition, and they are listed here for completeness. On the other hand, $\mathcal{RW}$ and $\mathcal{A}$ are different. A detailed characterization of the R/W-RAS has as follows:

1. $\mathcal{R} = \{R_1, \ldots, R_\mu\}$ is the set of the *(conventional) resources*.

2. $\mathcal{RW} = \{RW_1, \ldots, RW_h\}$ is the set of the *reader/writer (R/W) resources*, and h is the cardinality of the set.

3. $C : \mathcal{R} \to \mathbb{Z}^+$ is the system capacity function for the conventional resource types.

4. $\mathcal{P} = \{J_1, \ldots, J_\zeta\}$ is the set of the system process types supported by the considered system configuration. Each process type J_j is a composite element itself; in particular, $J_j = \langle \mathcal{S}_j, \mathcal{G}_j \rangle$, where: (a) $\mathcal{S}_j = \{\Xi_{j1}, \ldots, \Xi_{j,\varpi(j)}\}$ is the set of processing stages involved in the definition of process type J_j, and (b) $\mathcal{G}_j$ is a connected *acyclic* digraph $(\mathcal{V}_j, \mathcal{E}_j)$ that defines the sequential logic of process type J_j, $j = 1, \ldots, \zeta$. More specifically, the node set $\mathcal{V}_j$ of graph $\mathcal{G}_j$ is in one-to-one correspondence with the processing stage set, $\mathcal{S}_j$, and furthermore, there are two subsets $\mathcal{V}_j^{\nearrow}$ and $\mathcal{V}_j^{\searrow}$ of $\mathcal{V}_j$ respectively defining the sets of initiating and terminating processing stages for process type J_j. The connectivity of digraph $\mathcal{G}_j$ is such that every node $v \in \mathcal{V}_j$ is accessible from the node set $\mathcal{V}_j^{\nearrow}$ and co-accessible to the node set $\mathcal{V}_j^{\searrow}$. Finally, any directed path of $\mathcal{G}_j$ leading from a node of $\mathcal{V}_j^{\nearrow}$ to a node of $\mathcal{V}_j^{\searrow}$ constitutes a complete execution sequence for process type J_j.

5. $\mathcal{A} : \bigcup_{j=1}^{\zeta} \mathcal{S}_j \rightarrow \prod_{i=1}^{\mu} \{0, \ldots, C_i\} \times \prod_{i=1}^{h} \{0, 1, 2\}$ is the *resource allocation function*, which associates every processing stage Ξ_{jk} with a *resource allocation request* $\mathcal{A}(j,k) \equiv \mathcal{A}_{jk}$. More specifically, each $\mathcal{A}_{jk}$ is an $\mu + h$-dimensional vector such that:

   - $\forall i = 1 : \mu,\ \mathcal{A}_{jk}[i]$ indicates the number of resource units of resource type R_i necessary to support the execution of stage Ξ_{jk}.
   - $\forall i = 1 : h,\ \mathcal{A}_{jk}[\mu + i]$ equals 1 *iff* Ξ_{jk} acquires RW_i in the reading mode, whereas $\mathcal{A}_{jk}[\mu + i]$ equals 2 *iff* Ξ_{jk} acquires RW_i in the writing mode. Otherwise $\mathcal{A}_{jk}[\mu + i]$ equals 0.

A key assumption in the considered R/W-RAS, that is implied by item #4 above, is the acyclic structure of all the process types. On the other hand, similar to the conventional RAS, it is assumed that $\mathcal{A}_{jk} \neq \mathbf{0}$, i.e., every processing stage requires at least one resource unit for its execution. According to the applied resource allocation

protocol, a process instance executing processing stage Ξ_{jk} will be able to advance to a successor processing stage $\Xi_{j,k+1}$, only after it is allocated the resource differential $(\mathcal{A}_{j,k+1}[i] - \mathcal{A}_{jk}[i])^+, \forall i \in \{1,\ldots,\mu+h\}$, and it is only upon this advancement that the process will release the resource units $|(\mathcal{A}_{j,k+1}[i] - \mathcal{A}_{jk}[i])^-|, \forall i \in \{1,\ldots,\mu+h\}$. Some further qualifications are necessary in order to specify completely the considered resource allocation protocol w.r.t. the allocation of the R/W resource types. In particular: (i) A process is allowed access to resource RW_i in the reading mode only if no other process is currently accessing RW_i in the writing mode. (ii) A process is allowed to access RW_i in the writing mode only if no other process is accessing RW_i in either the reading or the writing modes. A process is also allowed to change its mode of accessing a resource RW_i as long as the two aforementioned rules are respected. Hence, if $\mathcal{A}_{jk}[\mu+i] = 1$ and $\mathcal{A}_{j,k+1}[\mu+i] = 2$, then the process will be able to advance from stage Ξ_{jk} to stage $\Xi_{j,k+1}$ only if no other process is concurrently accessing RW_i in the reading mode. On the other hand, if $\mathcal{A}_{jk}[\mu+i] = 2$ and $\mathcal{A}_{j,k+1}[\mu+i] = 1$, then the process can advance immediately from stage Ξ_{jk} to stage $\Xi_{j,k+1}$, changing the mode of acquisition of RW_i from writing to reading, when the allocation protocol constraints pertinent to the other resource types are satisfied.

Similar to the conventional RAS, the number of distinct processing stages supported by the considered R/W-RAS shall be denoted by ξ. Also, $\eta_{kl},\ k = 1,\ldots,\mu,\ l = 1,\ldots,C_k$, will denote the number of processing stages that require the allocation of l units from resource type R_k, whereas $\eta_{\mu+k,1}$ (resp., $\eta_{\mu+k,2}$), $k = 1,\ldots,h$ will denote the number of processing stages accessing RW_k in the reading (resp., writing) mode. Finally, in the sequel, unless qualified otherwise, any reference to a *resource* will imply any member of the set $\mathcal{R} \cup \mathcal{RW}$.

Modeling the dynamics of the R/W-RAS as a State Automaton: The dynamics of the R/W-RAS $\Phi_{rw} = \langle \mathcal{R}, \mathcal{RW}, C, \mathcal{P}, \mathcal{A} \rangle$, introduced in the previous paragraph, can be formally described by a Deterministic State Automaton (DSA), $G(\Phi_{rw})$

$= \langle S, E, f, \Gamma, \mathbf{s}_0, S_M \rangle$. E, f, Γ, $\mathbf{s}_0$, and S_M are identical to their counterpart in the conventional RAS (c.f. Section 1.2). On the other hand, the state set S consists of ξ-dimensional vectors $\mathbf{s}$ such that the components $\mathbf{s}[q]$, $q = 1, \ldots, \xi$, of $\mathbf{s}$ are in one-to-one correspondence with the R/W-RAS processing stages, and they indicate the number of process instances executing the corresponding stage in the R/W-RAS state modeled by $\mathbf{s}$. Hence, S consists of all the vectors $\mathbf{s} \in (\mathbb{Z}_0^+)^\xi$ that further satisfy

$$\forall i := 1 \ldots \mu, \quad \sum_{q=1}^{\xi} \mathbf{s}[q] \cdot \mathcal{A}(\Xi_q)[i] \leq C_i \tag{7.1}$$

$$\begin{aligned} \forall i := 1 \ldots h, \ (\exists q' : \ \mathbf{s}[q'] > 0 \ \wedge (\mathcal{A}(\Xi_{q'})[\mu + i]) = 2) \\ \Longrightarrow \sum_{q=1}^{\xi} \mathbf{s}[q] \cdot (\mathcal{A}(\Xi_q)[\mu + i]) = 2 \end{aligned} \tag{7.2}$$

Constraint 7.1 expresses the capacity constraints that must be observed w.r.t. the conventional resources. Constraint 7.2 enforces the exclusive acquisition of a R/W resource in the writing mode. In particular, at a state $\mathbf{s}$, if there exists an active process at processing stage q' and q' involves accessing RW_i in the writing mode, then any other processing stage that accesses RW_i in either the writing or the reading mode must have zero process content at $\mathbf{s}$. In the sequel, the number of units from resource R_i that are allocated to some active process instance in state $\mathbf{s}$ will be denoted by $\mathbf{s}.R_i$. On the other hand, for a R/W resource RW_i, $\mathbf{s}.RW_i = 1$ (resp., 2) *iff* RW_i is accessed in state $\mathbf{s}$ in the reading (resp., writing) mode by a single process, and $\mathbf{s}.RW_i = 3$ *iff* RW_i is accessed in the reading mode by multiple processes; otherwise, $\mathbf{s}.RW_i = 0$.

It is important to note that if a processing stage involves only the access of R/W resources in the reading mode, and no further allocation of any conventional resources, then an arbitrary number of processes might exist simultaneously in this stage, a fact that implies that the state space of the automaton might be infinite.

The target behavior of $G(\Phi_{rw})$ and the maximally permissive DAP: The sets S_r, S_s, $S_{\bar{s}}$, $S_{\bar{r}}$, S_{rs}, $S_{r\bar{s}}$, S_d, and S_{rd} are identical to their counterparts that were defined for conventional RAS. Hence, a maximally permissive deadlock avoidance policy (DAP) for R/W-RAS Φ_{rw} is a supervisory control policy that restricts the system operation within the subspace S_{rs}, guaranteeing, thus, that every initiated process can complete successfully. This definition further implies that the maximally permissive DAP is unique and can be implemented by an one-step-lookahead mechanism that recognizes and prevents transitions to unsafe states. On the other hand, due to the potentially infinite nature of the state space of R/W-RAS, the computation of the maximally permissive DAP cannot be performed through the basic enumerative techniques that are amenable for the conventional RAS studied in the previous chapters. Therefore, the technique proposed in this chapter is of paramount importance for the effective deployment of the maximally permissive deadlock avoidance in the considered RAS context.

Some monotonicities observed by the state unsafety concept: The monotonicity property defined in Section 2.5 for conventional RAS applies for R/W-RAS, and it is repeated here for emphasis

Proposition 7.1 *Consider a R/W-RAS Φ_{rw} and its states $\mathbf{s}$, $\mathbf{s}'$. Then,*

$$\mathbf{s} \in S_{\bar{s}} \ \wedge \ \mathbf{s} \preceq \mathbf{s}' \quad \Longrightarrow \quad \mathbf{s}' \in S_{\bar{s}}$$

Finally, the set of minimal reachable unsafe states $\hat{S}_{r\bar{s}}$ and the set of minimal reachable deadlock states $\hat{S}_{rd}$ are identical to their counterparts defined for conventional RAS.

7.3 On the finiteness and computation of $\hat{S}_{r\bar{s}}$

It is easy to see that $\hat{S}_{r\bar{s}}$ is a set of incomparable vectors w.r.t. the partial order '$\preceq$' defined in Equation 2.1. That is, $\forall \mathbf{x}, \mathbf{x}' \in \hat{S}_{r\bar{s}}, \ \mathbf{x} \not\succeq \mathbf{x}' \ \wedge \ \mathbf{x} \not\preceq \mathbf{x}'$. Therefore, by Dickson's Lemma (c.f. [20], Lemma 2A), $\hat{S}_{r\bar{s}}$ is a finite set.

A set U over ξ-dimensional vectors of natural numbers is called "*upward-closed*" (or "*right-closed*") if $\forall \mathbf{x} \in U,\ \mathbf{y} \succeq \mathbf{x} \Longrightarrow \mathbf{y} \in U$. Proposition 7.1 implies that the set $S_{r\bar{s}}$ resembles the structure of upward-closed sets; however, because its elements must satisfy the resource feasibility constraints (Eqs. 7.1-7.2) and they must also be accessible from $\mathbf{s}_0$, $S_{r\bar{s}}$ is not upward-closed, in a strict sense. To circumvent the technical difficulties arising from this fact, we define the set $U(S_{r\bar{s}}) \equiv S_{r\bar{s}} \cup \{\mathbf{y} \in \mathbb{N}^{\xi} \mid \exists \mathbf{x} \in S_{r\bar{s}} \text{ s.t. } \mathbf{y} \succ \mathbf{x}\}$. It can be easily seen that $U(S_{r\bar{s}})$ is upward-closed, and that it shares the same set of minimal elements with $S_{r\bar{s}}$.

Let $\mathbb{N}_{\omega} \equiv \mathbb{N} \cup \{\omega\}$, where the element ω denotes an arbitrarily large number; in particular, $\forall n \in \mathbb{N},\ \max\{n,\omega\} = \omega$ and $\min\{n,\omega\} = n$. Also, for any $\mathbf{x} \in \mathbb{N}_{\omega}^{\xi}$, define $reg(\mathbf{x}) \equiv \{\mathbf{x}' \in \mathbb{N}^{\xi} : \mathbf{x}' \preceq \mathbf{x}\}$. In [83], it is established that the minimal elements of a right-closed set U are effectively computable *iff* the decision problem '$(reg(\mathbf{x}) \cap U \neq \emptyset)$?' is decidable for every $\mathbf{x} \in \mathbb{N}_{\omega}^{\xi}$. Also, that work provides an algorithm for the enumeration of the set of minimal elements of a right-closed set U, when the test '$(reg(\mathbf{x}) \cap U \neq \emptyset)$?' is effectively computable (c.f. Theorem 2.14 in that work). In the light of that result, a potential approach to effectively construct $\hat{S}_{r\bar{s}}$, is by trying first to develop an algorithm for the resolution of the test '$(reg(\mathbf{x}) \cap U(S_{r\bar{s}}) \neq \emptyset)$?' for every $\mathbf{x} \in \mathbb{N}_{\omega}^{\xi}$, and subsequently apply the algorithm of [83]. Furthermore, it is easy to see that as long as $\mathbf{x}$ remains in $\mathbb{N}^{\xi}$, the question '$(reg(\mathbf{x}) \cap U(S_{r\bar{s}}) \neq \emptyset)$?' can be effectively resolved by constructing the subspace of S that is contained in $reg(\mathbf{x})$ and assessing the safety of every state in that subspace. The main challenge regarding the resolution of the aforementioned test is in the case that the considered vector $\mathbf{x}$ contains ω elements, especially in the state coordinates that can be arbitrarily large. These more complicated cases can be potentially resolved by developing an upper bound for the number of process instances that can execute simultaneously any of the RAS stages in a minimal reachable unsafe state. In fact, the availability of such a bound B can enable an even more straightforward algorithm for the enumeration of

$\hat{S}_{r\bar{s}}$ than the aforementioned algorithm in [83]: One could just construct the partial state space contained in $reg([B, B, \ldots, B]^T)$ and identify the set of minimal reachable unsafe states in that subspace; this set would constitute the entire $\hat{S}_{r\bar{s}}$.

However, in this chapter we customize and extend the methodology developed in Chapter 6 for the efficient enumeration of minimal reachable unsafe states. This methodology is motivated and enabled by the fact that, due to the acyclic structure of the process types in the considered R/W-RAS, unsafety is defined by unavoidable absorption into the system deadlocks. Hence, the unsafe states of interest can be retrieved by a localized computation that starts from the R/W-RAS deadlocks and "backtraces" the R/W-RAS dynamics until it hits the boundary between safe and unsafe subspaces. In particular, our interest in minimal reachable unsafe states implies that we can focus this backtracing only to minimal reachable deadlocks. The resulting algorithm decomposes naturally into a two-stage computation, with the first stage identifying all minimal reachable deadlocks, and the second stage performing the aforementioned backtracing process in order to identify the broader set of minimal reachable unsafe states. This algorithm will be the content of the next section.

7.4 Enumerating $\hat{S}_{r\bar{s}}$

Given a R/W-RAS ϕ_{rw}, the characterization of the minimal reachable deadlock states and the minimal reachable unsafe states is parallel to the characterization of the corresponding sets for the conventional RAS. Hence, the first-stage algorithm developed in Section 6.2 to enumerate the minimal reachable deadlock states shall be extended in this section to encompass the dynamics of the R/W resources. On the other hand, the definition of unsafety mentioned in the previous section and the implied properties are common for conventional RAS and R/W-RAS. Therefore, the developments of the second-stage algorithm are exactly similar to those of Section 6.3, and hence, the reader is referred to that section for a formal statement of the algorithm, and for a formal correctness analysis. However, in Section 6.3, the convergence of the

presented algorithm was established on the finiteness of the underlying state space, an assumption that does not hold any more for R/W-RAS. Therefore, the algorithm convergence is revisited in Subsection 7.4.3.

Based on the above remarks, the rest of this section proceeds as follows: First, we extend some terms and notation that were defined in Chapter 6 to address the deadlock formation in R/W-RAS. Next, we proceed to describe the flow of the first-stage algorithm. Finally, we prove the convergence of the second-stage algorithm in the context of R/W-RAS. To smoothen the flow of the exposition, and for self-containment, some of the content of Section 6.2 is recapitulated here.

7.4.1 Preamble

Given an edge $e \in \mathcal{G}$, e is "*blocked*" at state $\mathbf{s}$ *iff* $\mathbf{s}[e.src] > 0$, and (i) $\exists R_k \in \mathcal{R}$ s.t. $\mathbf{s}.R_k + \mathcal{A}_{e.dst}[k] - \mathcal{A}_{e.src}[k] > C_k$, or (ii) $\exists RW_k \in \mathcal{RW}$ s.t. $\mathcal{A}_{e.src}[\mu + k] = 0$, $\mathcal{A}_{e.dst}[\mu + k] = 2$, and $\mathbf{s}.RW_k > 0$, or (iii) $\exists RW_k \in \mathcal{RW}$ s.t. $\mathcal{A}_{e.src}[\mu + k] = 0$, $\mathcal{A}_{e.dst}[\mu + k] = 1$, and $\mathbf{s}.RW_k = 2$, or (iv) $\exists RW_k \in \mathcal{RW}$ s.t. $\mathcal{A}_{e.src}[\mu + k] = 1$, $\mathcal{A}_{e.dst}[\mu + k] = 2$, and $\mathbf{s}.RW_k = 3$.

In plain terms, an edge e is blocked at state $\mathbf{s}$ by a conventional resource if the slack capacity is less than the required number of additional units needed for $e.dst$. Furthermore, e is blocked by an R/W resource type RW_k if (a) $e.src$ does not access RW_k, RW_k is accessed in any of its modes at $\mathbf{s}$, and $e.dst$ requests RW_k in the writing mode, or (b) $e.src$ does not access RW_k, RW_k is accessed in the writing mode at $\mathbf{s}$, and $e.dst$ requests RW_k in the reading mode, or (c) $e.src$ accesses RW_k in the reading mode, RW_k is accessed by multiple processes in the reading mode at $\mathbf{s}$, and $e.dst$ requests RW_k in the writing mode.

On the other hand, e is "*enabled*" *iff* $\mathbf{s}[e.src] > 0$ and e is not blocked. Similarly, a processing stage q is blocked at state $\mathbf{s}$ *iff* all its outgoing edges are blocked, whereas it is enabled *iff* at least one of its outgoing edges is enabled. The set of enabled

edges at $\mathbf{s}$ shall be denoted by $g(\mathbf{s})$. Finally, we remind the reader that according to Definition 6.1, λ_X (the combination of the set of states X) is defined as $\lambda_X[q] \equiv max_{\mathbf{x} \in X} \mathbf{x}[q],\ q = 1 \ldots \xi$.

7.4.2 Enumerating $\hat{S}_{rd}$

By its definition, a minimal deadlock state $\mathbf{s}_d$ is a state at which all its processing stages with non-zero process content are blocked. Let $\{q_1, \ldots, q_t\}$ refer to this set of processing stages, and let $\{e_{i1}, \ldots, e_{iw}\}$ refer to the set of edges emanating from q_i. Then , following the rationale of Lemma 6.1, we can see that a minimal deadlock state $\mathbf{s}_d$ with active processing stages $\{q_1, \ldots, q_t\}$ can be expressed as the combination of a set of minimal states $\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}$ such that q_i is blocked at $\mathbf{x}_i$, i.e., $\mathbf{s}_d = \lambda_{\{\mathbf{x}_1, \ldots, \mathbf{x}_t\}}$. Similarly, we can perceive $\mathbf{x}_i$ as a combination of a set of minimal states $\{\mathbf{x}_{i1}, \ldots, \mathbf{x}_{iw}\}$ such that e_{ij} is blocked at state $\mathbf{x}_{ij}$, i.e., $\mathbf{x}_i = \lambda_{\{\mathbf{x}_{i1}, \ldots, \mathbf{x}_{iw}\}}$. Each $\mathbf{x}_{ij}$ is a state that has active processes at stage $e_{ij}.src$, and (i) for some resource type R_k s.t. $\mathcal{A}_{e_{ij}.dst}[k] - \mathcal{A}_{e_{ij}.src}[k] > 0$, $\mathbf{x}_{ij}.R_k > C_k - \mathcal{A}_{e_{ij}.dst}[k] + \mathcal{A}_{e_{ij}.src}[k] \equiv l$, or (ii) for some R/W resource type RW_k s.t. $\mathcal{A}_{e_{ij}.src}[\mu + k] = 0,\ \mathcal{A}_{e_{ij}.dst}[\mu + k] = 2,\ \mathbf{x}_{ij}.RW_k = 1\ \vee\ \mathbf{x}_{ij}.RW_k = 2$, or (iii) for some R/W resource type RW_k s.t. $\mathcal{A}_{e_{ij}.src}[\mu + k] = 0,\ \mathcal{A}_{e_{ij}.dst}[\mu + k] = 1,\ \mathbf{x}_{ij}.RW_k = 2$, or (iv) for some R/W resource type RW_k s.t. $\mathcal{A}_{e_{ij}.src}[\mu + k] = 1,\ \mathcal{A}_{e_{ij}.dst}[\mu + k] = 2,\ \mathbf{x}_{ij}.RW_k = 3$. Hence, the minimal states that block e_{ij} through R_k can be obtained by enumerating all the minimal states that allocate $l+1, \ldots, C_k$ units of R_k, and the minimal states that block e_{ij} through RW_k can be obtained by enumerating all the minimal states at which $\mathbf{s}.RW_k = 1,\ 2,\ 3$. A minimal state at which $\mathbf{s}.RW_k = 1 (resp., 2)$ is a single unit vector with one at a processing stage that accesses RW_k in the reading (resp., writing) mode. On the other hand, a minimal state at which $\mathbf{s}.RW_k = 3$ is a state that has exactly two processes executing a pair of processing stages that accesses RW_k in the reading mode.

Outline of the proposed algorithm: The proposed algorithm is motivated by the analysis presented in the pervious paragraph, and it can be described as follows:

1. For each resource type R_k, and for each occupancy level l, $1 \leq l \leq C_k$, compute the set of minimal states that allocate l units of R_k; call it $MinStR[k][l]$.

2. For each R/W resource type RW_k, compute the set of minimal states at which RW_k is acquired in either the reading or the writing modes; these sets are denoted respectively by $MinStR[\mu+k][1]$ and $MinStR[\mu+k][2]$.

3. Use the results obtained in Steps 1–2 in order to compute, for each edge e, the set of minimal states at which e is blocked; call it $BlockEd[e]$.

4. Use the results obtained in Step 3 in order to compute, for each processing stage q, the set of minimal states at which q is blocked; call it $BlockPs[q]$.

5. Finally, enumerate the set of minimal deadlocks through the following recursive scheme that, for each processing stage q and each minimal state $\mathbf{s} \in BlockPs[q]$, does the following: It sets $\mathbf{p}_1 := \mathbf{s}$, and then searches for an enabled processing stage q' at $\mathbf{p}_1$. Next, it branches for each minimal state $\mathbf{x}$ at which q' is blocked (i.e., $\mathbf{x} \in BlockPs[q']$), combining such a state with $\mathbf{p}_1$ (i.e., it computes the combination $\lambda_{\{\mathbf{p}_1,\mathbf{x}\}}$). Let $\mathbf{p}_2$ be a (feasible) state generated at one of those branches; i.e., $\mathbf{p}_2 = \lambda_{\{\mathbf{p}_1,\mathbf{x}'\}}$, $\mathbf{x}' \in BlockPs[q']$. State $\mathbf{p}_2$ is processed in a similar manner with state $\mathbf{p}_1$ above, and the branching continues across all the generated paths of the resulting search graph until a deadlock state is reached on each path.

As it can be revealed from the above discussion, Steps 1, 4, and 5 are exactly similar to their counterparts in Chapter 6 (i.e., Procedures 6.1, 6.3, and 6.4). Therefore, the reader is referred to Section 6.2 for the details of these procedures. On the other

hand, we need to illustrate Step 2 and extend Procedure 6.2 to support Step 3 in the context of R/W-RAS. This will be the content of the rest of this subsection.

Computing $MinStR[\mu+k][1]$ and $MinStR[\mu+k][2]$ for RW_k: As already pointed out, a minimal state $\mathbf{s}$ at which RW_k is accessed by a single process in the reading (resp., writing) mode can only be a unit vector state $\mathbf{s}[q]=1$, $\mathbf{s}[q']=0$, $\forall q \neq q'$, such that $\mathcal{A}_q[\mu+k]=1$ (resp., $\mathcal{A}_q[\mu+k]=2$). Therefore, $MinStR[\mu+k][1]$ (resp., $MinStR[\mu+k][2]$) shall contain only the $\eta_{\mu+k,1}$ (resp., $\eta_{\mu+k,2}$) unit vector states corresponding to the stages that access RW_k in the reading (resp., writing) mode.

Computing $BlockEd[e]$: According to the remarks that were provided in the beginning of this subsection, the computation of this data structure can be organized as follows: For each conventional resource R_k s.t. $\mathcal{A}_{e.dst}[k]-\mathcal{A}_{e.src}[k]>0$, and for each occupancy level l s.t. $l>C_k-\mathcal{A}_{e.dst}[k]+\mathcal{A}_{e.src}[k]$, we insert all the states from $MinStR[k][l]$ into $BlockEd[e]$ after adding one process at $e.src$, if needed. On the other hand, for each R/W resource RW_k s.t. $\mathcal{A}_{e.src}[\mu+k]=0$, $\mathcal{A}_{e.dst}[\mu+k]=1$, we insert all the states from $MinStR[\mu+k][2]$ into $BlockEd[e]$, whereas for each R/W resource RW_k s.t. $\mathcal{A}_{e.src}[\mu+k]=0$, $\mathcal{A}_{e.dst}[\mu+k]=2$, we insert all the states from $MinStR[\mu+k][1] \cup MinStR[\mu+k][2]$ into $BlockEd[e]$; we also add one process at $e.src$ in both cases. Finally, for each R/W resource RW_k s.t. $\mathcal{A}_{e.src}[\mu+k]=1$, $\mathcal{A}_{e.dst}[\mu+k]=2$, we insert all the states from $MinStR[\mu+k][1]$ after adding one process at $e.src$. The complete algorithm supporting this computation is depicted in Procedure 7.1.

The complexity analysis presented for Procedure 6.2 establishes that the running time of the first μ iterations of Procedure 7.1, dealing with conventional resource types, is no more than $\mathcal{O}(\sum_{k=1}^{\mu}\eta_k{}^{C_k})$. On the other hand, since $MinStR[k][l], k=\mu+1 \rightarrow \mu+h$, $l=1 \rightarrow 2$ contains only η_{kl} states, the running time of the last h iterations, dealing with R/W resource types, is no more than $\mathcal{O}\sum_{k=\mu+1}^{\mu+h}(\eta_{k1}+\eta_{k2})$. Therefore, the overall complexity of Procedure 7.1 is $\mathcal{O}((\mu+h)\cdot\xi\cdot(\sum_{k=1}^{\mu}\eta_k{}^{C_k}+\sum_{k=\mu+1}^{\mu+h}(\eta_{k1}+$

Procedure 7.1 CompBlockEd(e)

Input: an edge e from the "union" process graph $\mathcal{G}$
Output: $BlockEd[e]$

```
1: for k := 1 → μ + h do
2:    startIdx ← 1, endIdx ← 0
3:    if k ≤ μ then
4:       if A_{e.dst}[k] − A_{e.src}[k] > 0 then
5:          startIdx ← C_k − A_{e.dst}[k] + A_{e.src}[k] + 1; endIdx ← C_k;
6:       end if
7:    else
8:       if A_{e.dst}[k] = 2 ∧ A_{e.src}[k] = 0 then
9:          startIdx ← 1; endIdx ← 2
10:      else if A_{e.dst}[k] = 2 ∧ A_{e.src}[k] = 1 then
11:         startIdx ← 1; endIdx ← 1
12:      else if A_{e.dst}[k] = 1 ∧ A_{e.src}[k] = 0) then
13:         startIdx ← 2; endIdx ← 2;
14:      end if
15:   end if
16:   for l = startIdx → endIdx do
17:      for s ∈ MinStR[k][l] do
18:         s' ← s
19:         if s'[e.src] = 0 then
20:            s'[e.src] ← 1
21:            if Feasible(s') then
22:               Insert s' into BlockEd[e]
23:            end if
24:         else
25:            Insert s' into BlockEd[e]
26:         end if
27:      end for
28:   end for
29: end for
30: return BlockEd[e]
```

η_{k2}))), where $\mathcal{O}((\mu+h)\cdot\xi)$ is the complexity of checking the feasibility of a state.

Concluding this subsection, we would like to remark that, similar to the conventional RAS, given a R/W-RAS, a process instance executing a terminal processing stage can immediately exit the system upon completion; hence, terminal processing stages do not have any active process instances at any minimal deadlock state. Therefore, these stages are ignored by the proposed algorithm.

7.4.3 Proving the convergence of Algorithm 6.7 in the context of R/W-RAS

As it was pointed out in the introduction of this section, the convergence of Algorithm 6.7 in the context of the conventional RAS was based on the finiteness of the underlying state space, an assumption that does not hold any more for the considered R/W-RAS. The next proposition establishes the sought termination. We remind the reader that the algorithm employs two lists: $\dot{U}$ to store backtraced unsafe states, and Q to store untraced unsafe states. Whenever a new unsafe state is inserted into either of them, $L \equiv \dot{U} \cup Q$ is checked for dominance, and it is updated accordingly. In particular, a state is deleted from L only if it dominates a newly generated state. Thus, it can be inferred that such a state never enters L again. Furthermore, a newly generated state is discarded if it dominates a state vector in L. These remarks are important for establishing the following proposition:

Proposition 7.2 *When applied on a R/W-RAS instance Φ_{rw}, Algorithm 6.7 terminates in a finite number of steps.*

Proof: First, we remind the reader that an unsafe state is discarded if it dominates some other state in $\dot{U} \cup Q$. Second, we notice that the algorithm will diverge only if it keeps adding states to Q forever. To trace the states added to Q, let L_1 and L_2 be two sets of states such that when a state $\mathbf{s}$ is added to Q, if $\mathbf{s}$ is incomparable with all the states in L_1, $\mathbf{s}$ is added to L_1. Otherwise, i.e., if $\mathbf{s}$ is dominated by some state in L_1, $\mathbf{s}$ is added to L_2. The third case, i.e., that $\mathbf{s}$ dominates some state in L_1, is ruled out by the opening remark in this proof. Thus, L_1 is a set of incomparable vectors, and by Dickson's Lemma, it can never grow to be an infinite set. But, if L_1 is a finite set, then L_2 will also be a finite set (because the number of states dominated by each state in L_1 is finite). Thus, the algorithm terminates in a finite number of steps. □

7.5 Example

In this section, we demonstrate our algorithms for minimal reachable unsafe state enumeration, using the R/W-RAS configuration defined in Table 7.1. The considered R/W-RAS has three conventional resource types, $\{R_1, R_2, R_3\}$, each with capacity $C_i = 1$, and one R/W resource type. It also has two process types, $\{J_1, J_2\}$, where each process type is a sequence of processing stages, each engaging a single resource type. For representational economy, a state will be represented by the set of processing stages with non-zero process content. $MinStR$ is depicted in Table 7.2. $BlockPs$ is depicted in Table 7.3. Since each processing stage q has only one outgoing edge $e_q \in \mathcal{G}_{i_q}$, $BlockPs[q] = BlockEd[e_q]$. As previously mentioned, terminal stages Ξ_{14} and Ξ_{25} are discarded from consideration. Figure 7.1 demonstrates the search tree for minimal deadlocks that is constructed by Procedure 6.4 during its iteration that starts from stage Ξ_{11} and state $\mathbf{ss}_1 = \{\Xi_{11}, \Xi_{12}\}$: To block Ξ_{12}, we can add the extra process content of the state in $BlockPs[\Xi_{12}] = \{\mathbf{ss}_4\}$, thus obtaining state $\{\Xi_{11}, \Xi_{12}, \Xi_{13}\}$. This expansion continues until deadlocks are found at each possible path in the search tree. The complete execution of Procedure 6.4 returned the first four states in Table 7.4 ($\{\mathbf{u}_1, \ldots, \mathbf{u}_4\}$).

Table 7.4 depicts the minimal reachable unsafe states obtained by applying Algorithm 6.7, and the result of backtracing from each of them. The boldfaced processing stages indicate the source nodes of the backtraced edge $\tau_{\mathbf{a}}$. Table 7.5 depicts the results of the application of the function *Combine*. Next, we describe the overall flow of Algorithm 6.7 in the context of this example. The algorithm starts by constructing and storing the four minimal reachable deadlocks $\mathbf{u}_1, \ldots, \mathbf{u}_4$. These states are traced back and the *Combine* function is applied to the resulting boundary safe states. Backtracing $\mathbf{u}_3$, $\mathbf{u}_5$, $\mathbf{u}_6$, and $\mathbf{u}_7$ generates $\mathbf{u}_5$, $\mathbf{u}_6$, $\mathbf{u}_7$, and $\mathbf{u}_8$ respectively.

Applying $Combine(\mathbf{a}_4, \dot{A})$ generates $\mathbf{u}_9$ which is traced back to generate $\mathbf{u}_{10}$. Finally $\mathbf{u}_{10}$ is traced back to generate $\mathbf{u}_{11}$. At this point, $\dot{A} = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4\}$ and $Q = \emptyset$.

Table 7.1: The R/W-RAS considered in Section 7.5

Resource Types:	$\{R_1, R_2, R_3\}$, $\{RW_1\}$
Resource Capacities:	$C_i = 1, \forall i \in \{1, \ldots, 3\}$
Process Type 1:	$read(RW_1) \rightarrow R_1 \rightarrow R_2 \rightarrow R_3$
Process Type 2:	$R_3 \rightarrow R_1 \rightarrow R_3 \rightarrow R_1 \rightarrow write(RW_1)$

Table 7.2: The array $MinStR$ for the R/W-RAS depicted in Table 7.1

$MinStR[1][1]$	$\{\Xi_{12}\}, \{\Xi_{22}\}, \{\Xi_{24}\}$
$MinStR[2][1]$	$\{\Xi_{13}\}$
$MinStR[3][1]$	$\{\Xi_{21}\}, \{\Xi_{23}\}$
$MinStR[4][1]$	$\{\Xi_{11}\}$
$MinStR[4][2]$	$\emptyset$

Table 7.3: The array $BlockPs$ for the R/W-RAS depicted in Table 7.1

$BlockPs[\Xi_{11}]$	$\mathbf{ss}_1 = \{\Xi_{11}, \Xi_{12}\}$, $\mathbf{ss}_2 = \{\Xi_{11}, \Xi_{22}\}$, $\mathbf{ss}_3 = \{\Xi_{11}, \Xi_{24}\}$
$BlockPs[\Xi_{12}]$	$\mathbf{ss}_4 = \{\Xi_{12}, \Xi_{13}\}$
$BlockPs[\Xi_{13}]$	$\mathbf{ss}_5 = \{\Xi_{13}, \Xi_{21}\}$, $\mathbf{ss}_6 = \{\Xi_{13}, \Xi_{23}\}$
$BlockPs[\Xi_{21}]$	$\mathbf{ss}_7 = \{\Xi_{21}, \Xi_{12}\}$, $\mathbf{ss}_8 = \{\Xi_{21}, \Xi_{22}\}$, $\mathbf{ss}_9 = \{\Xi_{21}, \Xi_{24}\}$
$BlockPs[\Xi_{22}]$	$\mathbf{ss}_{10} = \{\Xi_{22}, \Xi_{21}\}$, $\mathbf{ss}_{11} = \{\Xi_{22}, \Xi_{23}\}$
$BlockPs[\Xi_{23}]$	$\mathbf{ss}_{12} = \{\Xi_{23}, \Xi_{12}\}$, $\mathbf{ss}_{13} = \{\Xi_{23}, \Xi_{22}\}$, $\mathbf{ss}_{14} = \{\Xi_{23}, \Xi_{24}\}$
$BlockPs[\Xi_{24}]$	$\mathbf{ss}_{15} = \{\Xi_{24}, \Xi_{11}\}$

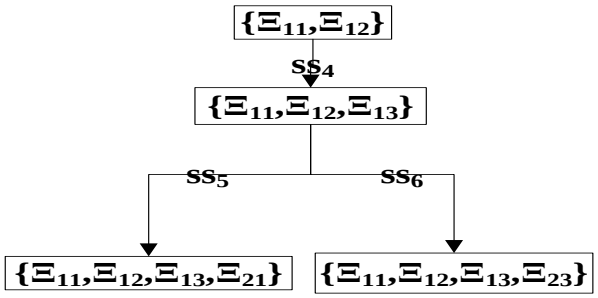


Figure 7.1: An iteration of Procedure 6.4 starting from $(\Xi_{11}, \mathbf{ss}_1)$

Hence, we proceed to Statement 15 to apply the $Confine$ operation, which does not generate any new minimal unsafe state, and the algorithm terminates.

Table 7.4: Tracing back from minimal unsafe states for the R/W-RAS depicted in Table 7.1

$\mathbf{u}$	$Prev\_Safe$	$Prev\_Unsafe$
$\mathbf{u}_1 = \{\Xi_{11}, \Xi_{24}\}$	$\mathbf{a}_1 = \{\Xi_{11}, \boldsymbol{\Xi_{23}}\}$	$\emptyset$
$\mathbf{u}_2 = \{\Xi_{12}, \Xi_{13}, \Xi_{21}\}$	$\mathbf{a}_2 = \{\boldsymbol{\Xi_{11}}, \Xi_{13}, \Xi_{21}\}$	$\emptyset$
$\mathbf{u}_3 = \{\Xi_{12}, \Xi_{13}, \Xi_{23}\}$	$\emptyset$	$\mathbf{u}_5 = \{\boldsymbol{\Xi_{11}}, \Xi_{13}, \Xi_{23}\}$
$\mathbf{u}_4 = \{\Xi_{21}, \Xi_{22}\}$	$\emptyset$	$\emptyset$
$\mathbf{u}_5 = \{\Xi_{11}, \Xi_{13}, \Xi_{23}\}$	$\mathbf{a}_3 = \{\Xi_{11}, \Xi_{13}, \boldsymbol{\Xi_{22}}\}$	$\mathbf{u}_6 = \{\Xi_{11}, \boldsymbol{\Xi_{12}}, \Xi_{23}\}$
$\mathbf{u}_6 = \{\Xi_{11}, \Xi_{12}, \Xi_{23}\}$	$\emptyset$	$\mathbf{u}_7 = \{\boldsymbol{2\Xi_{11}}, \Xi_{23}\}$
$\mathbf{u}_7 = \{2\Xi_{11}, \Xi_{23}\}$	$\emptyset$	$\mathbf{u}_8 = \{2\Xi_{11}, \boldsymbol{\Xi_{22}}\}$
$\mathbf{u}_8 = \{2\Xi_{11}, \Xi_{22}\}$	$\mathbf{a}_4 = \{2\Xi_{11}, \boldsymbol{\Xi_{21}}\}$	$\emptyset$
$\mathbf{u}_9 = \{2\Xi_{11}, \Xi_{21}, \Xi_{13}\}$	$\emptyset$	$\mathbf{u}_{10} = \{2\Xi_{11}, \Xi_{21}, \boldsymbol{\Xi_{12}}\}$
$\mathbf{u}_{10} = \{2\Xi_{11}, \Xi_{21}, \Xi_{12}\}$	$\emptyset$	$\mathbf{u}_{11} = \{\boldsymbol{3\Xi_{11}}, \Xi_{21}\}$
$\mathbf{u}_{11} = \{3\Xi_{11}, \Xi_{21}\}$	$\emptyset$	$\emptyset$

Table 7.5: The application of the "Combine" operation in the context of Algorithm 6.7 on the R/W-RAS depicted in Table 7.1

$\mathbf{a}$	*Combine* results	comment	$\dot{A}$
$\mathbf{a}_1$	–	–	$\{\mathbf{a}_1\}$
$\mathbf{a}_2$	$(\mathbf{a}_2, \mathbf{a}_1) = \{\Xi_{11}, \Xi_{21}, \Xi_{13}, \Xi_{23}\}$	infeasible	$\{\mathbf{a}_1, \mathbf{a}_2\}$
$\mathbf{a}_3$	$(\mathbf{a}_3, \mathbf{a}_1) = \{\Xi_{11}, \Xi_{22}, \Xi_{13}, \Xi_{23}\}$ $(\mathbf{a}_3, \mathbf{a}_2) = \{\Xi_{11}, \Xi_{21}, \Xi_{13}, \Xi_{22}\}$	dominates $\mathbf{u}_5$ dominates $\mathbf{u}_4$	$\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$
$\mathbf{a}_4$	$(\mathbf{a}_4, \mathbf{a}_1) = \{2\Xi_{11}, \Xi_{21}, \Xi_{23}\}$ $(\mathbf{a}_4, \mathbf{a}_2) = \{2\Xi_{11}, \Xi_{13}, \Xi_{21}\}$ $(\mathbf{a}_4, \mathbf{a}_3) = \{2\Xi_{11}, \Xi_{13}, \Xi_{21}, \Xi_{22}\}$	infeasible generates $\mathbf{u}_9$ dominates $\mathbf{u}_4$	$\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4\}$

7.6 Concluding remarks

This chapter has extended the definition of the RAS abstraction to encompass the dynamics of R/W-locks. It has also extended the methodology introduced in Chapter 6 for the enumeration of the set of minimal reachable unsafe states so that it applies to R/W-RAS, and has established the significance of this capability for the effective implementation of the maximally permissive DAP in the context of this new RAS class.

CHAPTER VIII

CONCLUSION AND FUTURE WORK

This work has developed a novel methodology for the effective deployment of the maximally permissive DAP in the context of the complex resource allocation that takes place in many contemporary applications. Our results have been enabled by (i) a careful distinction between the off-line and the on-line parts of the computation that is required for the effective characterization and deployment of the target policy, and (ii) the effective control of the complexity involved in the on-line part of the computation through the employment of pertinent representations and data structures. More specifically, under the proposed approach, the on-line implementation of the maximally permissive DAP is perceived as a classifier that effects the dichotomy of the underlying state space into the safe and unsafe subspaces, and therefore, the efficient implementation of the policy reduces to the synthesis of a classifier that can express the sought separation of the state space in a succinct and compact manner. It has also been shown that certain properties of the underlying state space enable extensive reductions of the information that must be explicitly considered during the classifier synthesis process, alleviating substantially the computational effort of the synthesis process. Moreover, we have proposed an algorithm that efficiently identifies and stores a critical subset of states for resolving the safety of the underlying RAS state space, while foregoing the complete enumeration of the state space. The cardinality of this critical set is smaller than that of the complete state space by many orders of magnitude.

The above ideas have received a thorough formal treatment in the context of

Gadara RAS and the broader RAS class of Definition 1.1. Furthermore, an implementation of the maximally permissive DAP for RAS with R/W locks has been proposed. At the same time, extensive numerical experimentation in the context of the aforementioned RAS classes has confirmed the tractability of the presented approach and its ability to provide parsimonious implementations of the maximally permissive DAP for RAS with a very large size and very large state spaces.

Future work

In this section, we discuss three lines of research that can extend the work presented in this thesis. The first line focuses on the combinatorial nature of the problem, in an effort to develop both theoretical and computational results, that can be employed for the construction of parametric classifiers. Alternatively, the second line of research shall address the construction of the parametric classifiers using methods and techniques borrowed from the statistical learning literature [85]. Finally, the third line of research shall seek to extend the capabilities of the efficient algorithms of Chapters 6 - 7.

The cornerstone of the first line of research mentioned in the previous paragraph is the connection between the parametric classifier design problem and the classical set-covering problem that has been studied in Operations Research and Computer Science. The aforementioned connection has been identified in Chapter 3. One can seek to explore and formalize further this connection, in an effort to (i) develop novel insights regarding the geometric and combinatorial structure of the parametric classifier design problem, and (ii) set an analytical base for the development of additional customized and computationally (more) efficient algorithms for its solution. An initial result along this direction was developed in [76]. Furthermore, to boost the scalability of the parametric classifiers, one can utilize the aforementioned problem affinity to the set covering problem to: (i) employ large-scale optimization methods for the

construction of the sought classifiers, and (ii) develop efficient branch & bound based techniques for the construction of the sought classifiers. An initial result along this direction was developed in [16].

A second line of research can seek the construction of the parametric classifiers using techniques and methods borrowed from the statistical learning literature. In the statistical learning methods, a sample of vectors from different classes is used to construct the sought classifier with some error tolerance, i.e., the synthesized classifier classifies the vectors in the selected sample with some predefined error. This sample of vectors are typically called the *training data set* in the relevant literature. The absence of classification error leads to overfitting the classifier to the sample used for its construction. On the other hand, the deployment of the maximally permissive DAP through a parametric classifier, as presented in this work, implies that (i) the training data set must contain all the possible vector realizations to which the system might evolve, and that (ii) the constructed classifier must have zero error tolerance. The aforementioned restrictions and the implied absence of randomness present some key special requirements that must be addressed in order to enable the employment of statistical learning methods for the construction of the sought classifiers. An additional problem that is largely overlooked in the statistical learning literature, is the development of systematic methods to reduce the size of the classifier. Hence, the minimization of the size of the sought classifier must also be addressed in this approach. Furthermore, as it has been shown in this thesis, we can significantly reduce the size of the training data set. However, this reduction imposes some restrictions on the structure of the sought classifiers. In conclusion, some important research questions that must be addressed by this research line are the following: Is it possible to extend the statistical learning algorithms to impose the aforementioned restrictions on the structure of the sought classifiers? How will these restrictions affect the performance of the algorithms? How will these restrictions affect the size of the constructed

classifier?

A third line of research shall seek to leverage the computational capabilities offered by the algorithms of Chapters 6 - 7, through the employment of Binary Decision Diagrams (BDDs) as a computational framework. The main advantage of BDDs is the computational efficiency that they provide through the symbolic computation of the target space. Hence, as mentioned in Chapter 1, the BDDs has been employed to synthesize the maximally permissive DAP through the symbolic computation of the reachable state space [1, 82, 48]. However, the aforementioned computation becomes unstable and inefficient, when the reachable state space is very large. On the other hand, using BDDs for the symbolic computation of the minimal unsafe states through the algorithms of Chapters 6 - 7 involves a search process that is restricted to a small proportion of the reachable state space. Hence, this restricted search offers possibilities for a significant enhancement of the capabilities of the aforementioned algorithms.

While the ideas described in the previous paragraph seek to extend the computational capabilities and to enhance the efficiency of the involved algorithms, another research direction might seek to extend the applicability of those algorithms, by enabling them to address RAS with cyclic process structures. The main difficulty in RAS with cyclic process structures is that the state unsafety can be attributed to either deadlocks or *livelocks*. Conceptually, a RAS livelock is defined by a set of states such that: (i) this set does not contain $\mathbf{s}_0$, i.e., the empty state, (ii) none of these states is a deadlock, and (iii) these states constitute a closed communicating class of the underlying state transition diagram, i.e., when the system reaches this set of states, it remains entrapped in it. To handle the existence of livelocks in the underlying the RAS behavior, the algorithms of Chapters 6 - 7 must enumerate the minimal deadlock states *and the minimal livelock states*. To this end, we shall seek to formally develop the necessary and sufficient conditions for the characterization of

the (minimal) livelock states, and subsequently, we shall seek to develop algorithmic procedures for their complete enumeration.

APPENDIX A

FINITE STATE AUTOMATA: SOME BASIC CONCEPTS AND DEFINITIONS

Among the class of qualitative behavioral models employed by Discrete Event System theory, the most straightforward, and, probably, the most widely used, is the *Finite State Automaton(FSA)* [9]. An excellent reference on the FSA model and theory can be found in [33]. A formal definition of this model is as follows:

Definition A.1 *[9] A* (Deterministic) Finite State Automaton (FSA) *G is a 6-tuple*

$$G = \langle S, E, f, \Gamma, s_0, S_m \rangle$$

where

- *S is a finite set, called the* state set *of the automaton;*
- *E is a finite set, called the* event set *of the automaton;*
- *$f: S \times E \to S$, is the* state transition function*, i.e., $\forall s \in S, \forall e \in E,\ f(s,e) = s'$ means that there is a transition from state s to state s' that is triggered by event e; in general, f is a partial function on its domain, i.e., certain events cannot occur in state s;*
- *$\Gamma : S \to 2^E$ is the* feasible event function*, i.e., $\forall s \in S,\ \Gamma(s)$ denotes the set of all events e for which $f(s,e)$ is defined;*
- *$s_0 \in S$ is the* initial state *of the automaton;*
- *$S_m \subseteq S$ is the set of* marked states*.*

The transitional structure expressed by the FSA model can be visualized by a labelled digraph $\mathcal{N}$; this directed graph is known as the *state transition diagram (STD)* of the FSA, and its node set corresponds to the state set S of the automaton, its edge set is defined by the state transition function, and its edge label set corresponds to the event set E of the automaton.

It is obvious from the above definitions that the FSA and its corresponding STD can be perceived as a complete map for the behavioral evolution of the modelled system. This effect can be formalized as follows:

Definition A.2 *[9] Consider an FSA* $G = \langle S, E, f, \Gamma, s_0, S_m \rangle$ *and let* E^* *denote the set containing all the* finite-length *sequences that can be generated from* E*, including the empty sequence* ϵ*.*

1. *The FSA state transition function* f *is naturally extended to* $S \times E^*$ *as follows:*

$$\forall s \in S, \quad f(s, \epsilon) \equiv s \tag{A.3}$$

$$\forall s \in S, \forall u \in E^*, \forall e \in E, \quad f(s, ue) \equiv f(f(s, u), e) \tag{A.4}$$

2. *The* language $L(G)$ generated *by* G *is defined by*

$$L(G) \equiv \{u \in E^* : f(s_0, u)!\} \ ^1 \tag{A.5}$$

3. *The* language $L_m(G)$ marked *by* G *is defined by*

$$L_m(G) \equiv \{u \in L(G) : f(s_0, u) \in S_m\} \tag{A.6}$$

In the STD context, $L(G)$ can be described as the set of all event sequences $u \in E^*$ that can be traced on any path, not necessarily simple, starting from the initial state s_0. $L_m(G)$ is used to model event sequences that correspond to the achievement of some "milestone" in the system behavior.

[1] *i.e.,* $f(s_0, u)$ *involves only transitions corresponding to feasible events, and therefore, it is well-defined*

APPENDIX B

AN EFFICIENT ALGORITHM FOR STATE SPACE ENUMERATION

Given a RAS $\Phi = \langle \mathcal{R}, C, \mathcal{P}, \mathcal{A} \rangle$, the solution framework for the construction of the maximally permissive DAP as depicted in Figure 2.1 requires, as a first step, the computation of the reachable state space S_r of the corresponding DFSA $G(\Phi)$. It is known that this step suffers from computational limitations due to the (very) large sizes of the set S_r. Furthermore, as it was remarked in the Chapter 1, this step corresponds to standard operations encountered in the R&W SC framework [9], and therefore, in principle, it can be performed by any procedure that has been developed in support of this operation. However, in practice, the applicability of some of these procedures might be challenged by the fact that we want to target RAS configurations with very large state spaces. In this appendix, we report a particular algorithm for the generation and storage of S_r that has been found to be especially efficient in our computational studies.

This algorithm provides an enumeration of S_r, by first identifying, as an intermediary step, all the states corresponding to a feasible resource allocation, according to the prevailing resource capacity constraints (c.f., Eq. 1.1); we shall refer to these RAS states as *"valid"* states, and the corresponding state set will be denoted by S_v. Once S_v has been constructed, a subsequent procedure filters out from it the set of reachable states S_r. Therefore, the whole computation is organized naturally into two major procedures: (a) that of generating state set S_v, and (b) that of reducing S_v to S_r. The introduction of the intermediate step of generating the state set S_v does not increase substantially the complexity of the involved computation, since, as

will be revealed in the subsequent discussion, both sets S_v and S_r are of comparable sizes. On the other hand, performing the overall computation through the proposed sequence enables a more efficient handling and storage of the information characterizing the underlying FSA structure, and also the effective utilization of the auxiliary memory in case that the involved data structures grow so big that they cannot be accommodated in the core memory.

B.1 Constructing the set of valid states S_v

To describe the first of the two procedures listed above, let us denote by K_{ij} the maximum number of process instances that can execute concurrently a processing stage Ξ_{ij} without violating the capacity restrictions imposed by the resources involved in the execution of this stage. In the following discussion we shall also use Ξ_{ij}^k to denote the existence of k active process instances at the processing stage Ξ_{ij}, and $\mathbf{s}_{ij}$ to denote the state component corresponding to processing stage Ξ_{ij}. Given a resource allocation state $\mathbf{s}$, we shall say that (the "process load" indicated by) $\Xi_{i'j'}^{k'}$ can be added to state $\mathbf{s}$ *iff* $\mathbf{s}_{i'j'} = 0$ and the state $\mathbf{s}' \equiv \{\forall (i,j) \neq (i',j') : \mathbf{s}'_{ij} = \mathbf{s}_{ij}$ and $\mathbf{s}'_{i'j'} = k'\}$ does not violate any resource capacity. The proposed algorithm enumerates the set of valid states, S_v, starting with state $\mathbf{s}_0 \equiv \mathbf{0}$, and subsequently considering for every generated state $\mathbf{s} \in S_v$, the possibility of adding $\Xi_{i'j'}^{k'}$ to it, for all i', j' and k'. This enumeration is systematized and facilitated by the following two data structures:

- A composite data structure called $Node_{\mathbf{s}}$, that supports the generation and processing of a single state $\mathbf{s}$ in the overall enumeration process. This data structure consists of the following two components:
  - $\mathbf{s}$: the vector representation of state $\mathbf{s}$.
  - $L_{\mathbf{s}}$: a list containing all the "process loads" Ξ_{ij}^k that can be added to state $\mathbf{s}$.

Algorithm B.1 The algorithm constructing the set of valid states S_v.

Input: Representation of a given resource allocation system Φ.

Output: The list of states that constitutes the valid state space S_v.

1: $S_v \leftarrow \emptyset$; $Q \leftarrow \emptyset$;
2: Insert $\mathbf{s}_0$ into S_v;
3: $L_{\mathbf{s}_0} := \{\Xi_{ij}^k : \forall = i \in \{1,\ldots,\zeta\}, \forall j \in \{1,\ldots,\varpi(i)\}, \forall k \in \{1,\ldots,K_{ij}\}$;
4: Add $Node_{\mathbf{s}_0} \equiv (\mathbf{s}_0, L_{\mathbf{s}_0})$ to Q;
5: **while** $Q! = \emptyset$ **do**
6: $Node_{\mathbf{s}} \equiv (\mathbf{s}, L_{\mathbf{s}}) \leftarrow$ Pop Q;
7: **for** each $\Xi_{pq}^r \in L_{\mathbf{s}}$ **do**
8: $\mathbf{s}^* \leftarrow$ Add$(\mathbf{s}, \Xi_{pq}^r)$;
9: $L_{\mathbf{s}^*} \leftarrow \{\Xi_{ij}^k : \Xi_{ij}^k$ can be added to state $\mathbf{s}^* \wedge ((i > p) \vee ((i = p) \wedge (j > q)))\}$;
10: Push $Node_{\mathbf{s}^*} \equiv (\mathbf{s}^*, L_{\mathbf{s}^*})$ to queue Q;
11: Insert $\mathbf{s}^*$ into S_v;
12: **end for**
13: **end while**
14: Return S_v

- The queue, Q, that contains all the state nodes that are waiting to be processed.

Processing a node $Node_{\mathbf{s}}$ implies (i) the generation of all the states $\mathbf{s}'$ that result from the addition to $\mathbf{s}$ of the loads included in the list L_s, (ii) the construction of the corresponding nodes $Node_{\mathbf{s}'}$, and (iii) the addition of these nodes to queue Q. The complete algorithm for generating the valid state space S_v is depicted in Algorithm B.1. Since every processing stage Ξ_{ij} can have up to K_{ij} active jobs, the list $L_{\mathbf{s}_0}$, constructed in Line 3, contains all the possible states that can be obtained from state $\mathbf{s}_0$ by activating only one processing stage, Ξ_{ij}, to some number of jobs in the interval $[1, K_{ij}]$. On the other hand, the while loop in Line 5, that processes the state nodes that are stored in queue Q, is broken down in the following steps: Line 6 extracts a node $(\mathbf{s}, L_{\mathbf{s}})$ stored in queue Q for further processing. For every element Ξ_{pq}^r in list $L_{\mathbf{s}}$, Line 8 constructs a new state $\mathbf{s}^*$ from Ξ_{pq}^r and $\mathbf{s}$ such that $\mathbf{s}^* = \{\forall (i,j) \neq (p,q) : \mathbf{s}_{ij}^* = \mathbf{s}_{ij}$ and $\mathbf{s}_{pq}^* = r\}$. Line 9, constructs the list $L_{\mathbf{s}^*}$ for the node $Node_{\mathbf{s}^*}$ corresponding to state s^* constructed in Line 8. This list contains all

Ξ_{ij}^k that satisfy the following conditions: First, load Ξ_{ij}^k can be added to state $\mathbf{s}^*$, i.e., (a) $\mathbf{s}_{ij}^* = 0$, and (b) adding the k jobs at stage Ξ_{ij} to all the active jobs at state $\mathbf{s}^*$ does not violate any resource capacities. Second, the index $(\sum_{a=0}^{i-1} \varpi(a) + j)$ of processing stage Ξ_{ij} in the state vector is strictly greater than the index $(\sum_{a=0}^{p-1} \varpi(a) + q)$ of the processing stage Ξ_{pq} in the state vector, which is true *iff* $(i > p) \vee ((i = p) \wedge (j > q))$. The first condition essentially filters the set of processing stages to detect those that can have active jobs concurrently with the active jobs in $\mathbf{s}^*$. The second condition is necessary in order to avoid the generation of a state more than once. Line 10 queues the constructed node $Node_{\mathbf{s}^*}$ for further processing. The loop is terminated when all the state nodes entered in queue Q have been processed. It should be clear from the above, that at this point, all the valid states have been generated.

Complexity analysis Line 6 as well as Lines 8 through 11 are executed $\mathcal{O}(|S_v|)$ times. On the other hand, the running time of Lines 6, 10, and 11 is $\mathcal{O}(1)$. The running time of Line 8 is $\mathcal{O}(\xi)$. The running time of Line 9 is $\mathcal{O}(\sum_{i=1}^{\zeta} \sum_{j=1}^{\varpi(i)} K_{ij})$. Therefore, the overall running time of the algorithm is $\mathcal{O}((\xi + \sum_{i=1}^{\zeta} \sum_{j=1}^{\varpi(i)} K_{ij}) \cdot |S_v|)$. Since, typically, $(\xi + \sum_{i=1}^{\zeta} \sum_{j=1}^{\varpi(i)} K_{ij}) \ll |S_v|$, we can say that the practical running time of the algorithm is $\mathcal{O}(|S_v|)$. We should emphasize that, while the aforestated result indicates a linear complexity of Algorithm B.1 with respect to $|S_v|$, S_v itself is, in general, exponentially sized with respect to the RAS size $|\Phi|$, and therefore, Algorithm B.1 remains an "expensive" computation. On the other hand, the established complexity of $\mathcal{O}(|S_v|)$ implies that Algorithm B.1 is an efficient algorithm for enumerating the set S_v (among all the algorithms that can support such an explicit enumeration).

To understand the efficiency of the proposed enumeration scheme, the reader should notice that whenever a state $\mathbf{s}$ is processed by Algorithm B.1, it is guaranteed that it will not be considered again. This treatment is essentially different from the treatment applied to the generated states by the standard search-type of algorithms

that are used for the direct enumeration of the reachable state space S_r. This last class of algorithms need to constantly check whether any newly reached state has been already generated, and this operation can be computationally demanding. It also necessitates a continuous access to the entire list of the generated states throughout the execution of those algorithms. On the other hand, by not revisiting a processed state, our algorithm does not need to keep such a state in the core memory, and therefore, processed states can simply be saved in a file on the hard disk.[1] This remark further implies that the memory consumption of the above algorithm is mainly due to the maintenance of the queue of unprocessed states, Q. But this consumption is quite controllable: whenever Q becomes relatively large, we can write some of the states in a file on the hard disk, remove them from the memory, process the rest of the states, and finally, re-load the saved file into the queue, and continue processing these additional states. Working in this way, we have been able to process RAS Φ with extremely large state spaces.

B.2 Extracting the set S_r from the set of valid states S_v

The second procedure that filters the set of valid states, S_v, to extract the set of reachable states, S_r, is presented in Algorithm B.2. In this procedure, L is a list of states, *reachableStack* is a stack of states, and *isReachable* is a binary array whose length equals the length of L, and such that $isReachable(i) = 1$ *iff* $L(i)$ is a reachable state. Then, it should be obvious that the depicted procedure implements a "reaching scheme" that marks all the reachable states in the provided set S_v, while starting from the initial state $\mathbf{s}_0$. In this reaching scheme, all the information regarding the state reachability is processed and stored through the binary array $isReachable()$, that is indexed by the previously generated listing of S_v, and, therefore, the memory footprint of the presented procedure remains quite efficient. Furthermore, the systematic

[1] Continuous writing on the hard disk is not encouraged though. So, we buffer the processed states, and write them to the hard disk in batches.

Algorithm B.2 The algorithm extracting the set S_r from the set of valid states S_v.

Input: The set of valid states S_v.

Output: The set of states that constitutes the reachable subspace S_r.

1: Initialize L with the elements of the input set S_v;
2: Sort L lexicographically in ascending order; {The empty state $\mathbf{s}_0$ will be the first state.}
3: $\forall i,\ isReachable(i) \leftarrow 0$; $reachableStack \leftarrow \emptyset$;
4: push $L(0)$ onto $reachableStack$; $isReachable(\mathbf{s}_0) := 1$;
5: **while** $reachableStack! = \emptyset$ **do**
6: $\quad \mathbf{s} \leftarrow$ pop $reachableStack$;
7: $\quad$ Identify all the events that can be executed from $\mathbf{s}$, and generate the corresponding list of its successor states, N_s;
8: $\quad$ **for** each state $\mathbf{s}' \in N_s$ **do**
9: $\quad\quad$ **if** $isReachable(\mathbf{s}') == 0$ **then**
10: $\quad\quad\quad$ push $\mathbf{s}'$ onto $reachableStack$;
11: $\quad\quad\quad$ $isReachable(\mathbf{s}') \leftarrow 1$;
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **end while**
15: $S_r := \{\mathbf{s} \in S_v : isReachable(\mathbf{s}) = 1\}$;
16: Return S_r

enumeration of the state set S_v established by Algorithm B.1, provides also a linear ordering for the elements of this set and an indexing scheme for direct accessing of the elements of the array $isReachable$, upon the provision of the corresponding state $\mathbf{s}$.

Complexity analysis Let $\bar{t}$ be the maximum number of transitions that emanate from any given state $\mathbf{s} \in S_r$. It is easy to see that $\bar{t}$ is upper-bounded by $\sum_{i=1}^{\zeta} |V_i^{\nearrow}| + |\mathcal{E}| + \sum_{i=1}^{\zeta} |V_i^{\searrow}|$, according to the notation introduced in Chapter 1, and therefore, it relates polynomially to the parameters defining the size of the underlying RAS. The while loop in Line 5 is executed $\mathcal{O}(|S_r|)$ times. Line 8 is executed $\mathcal{O}(\bar{t})$ times in a single iteration of the while loop. Checking the if-condition inside Line 8 upon any given state $\mathbf{s}'$ takes $\mathcal{O}(log(|S_r|))$ time, using binary search. So, the overall complexity of the above algorithm is $\mathcal{O}(\bar{t} \cdot |S_r| \cdot log(|S_r|))$. Since, typically $\bar{t} \ll |S_r|$, we can also say that the practical complexity of the considered algorithm is $\mathcal{O}(|S_r| \cdot log(|S_r|))$.

B.3 The overall complexity

Finally, combining the results regarding the computational complexities of Algorithms B.1 and B.2, and taking into consideration the additional fact that $|S_v| \approx \mathcal{O}(|S_r|)$, we can also infer that the practical complexity of the entire computation of the set S_r, according to the proposed scheme, is $\mathcal{O}(|S_r| \cdot log(|S_r|))$.

REFERENCES

[1] Akers, S., "Binary decision diagrams," *IEEE Transactions on Computers*, vol. 100, no. 6, pp. 509–516, 1978.

[2] Akesson, K., Fabian, M., Flordal, H., and Malik, R., "SUPREMICA-an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proceedings of the 8th International Workshop on Discrete Event Systems*, pp. 384–385, IEEE, 2006.

[3] Araki, T., Sugiyama, Y., and Kasami, T., "Complexity of the deadlock avoidance problem," in *Proceedings of the 2nd IBM Symposium on Mathematical Foundations of Computer Science*, pp. 229–257, 1977.

[4] Badouel, E. and Darondeau, P., "Theory of regions," in *LNCS 1491 – Advances in Petri Nets: Basic Models* (Reisig, W. and Rozenberg, G., eds.), pp. 529–586, Springer-Verlag, 1998.

[5] Banaszak, Z. A. and Krogh, B. H., "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows," *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 724–734, 1990.

[6] Barkaoui, K., Chaoui, A., and Zouari, B., "Supervisory control of discrete event systems based on structure theory of Petri nets," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pp. 3750–3755, 1997.

[7] Brass, P., *Advanced Data Structures*. NY,NY: Cambridge University Press, 2008.

[8] Bryant, R., "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 677–691, 1986.

[9] Cassandras, C. G. and Lafortune, S., *Introduction to Discrete Event Systems (2nd ed.)*. NY,NY: Springer, 2008.

[10] Chen, Y. and Lin, F., "Modeling of discrete event systems using finite state machines with parameters," *CCA00, Anchorage, Alaska*, 2000.

[11] Chu, F. and Xie, X.-L., "Deadlock analysis of Petri nets using siphons and mathematical programming," *IEEE Transactions on Robotics and Automation*, vol. 13, pp. 793–804, 1997.

[12] Ciardo, G., "Reachability set generation for Petri nets: Can brute force be smart?," *Applications and Theory of Petri Nets*, pp. 17–34, 2004.

[13] Clarke Jr., E. M., Grumberg, O., and Peled, D. A., *Model Checking*. Cambridge, MA: The MIT Press, 1999.

[14] Coffman, E. G., Elphick, M. J., and Shoshani, A., "System deadlocks," *Computing Surveys*, vol. 3, pp. 67–78, 1971.

[15] Commer, P. and Sethi, R., "The complexity of trie index construction," *Journal of the ACM*, vol. 24, pp. 428–440, 1977.

[16] Cordone, R., Nazeem, A., Piroddi, L., and Reveliotis, S., "Maximally permissive deadlock avoidance for sequential resource allocation systems using disjunctions of linear classifiers," in *Proceedings of CDC 2012*, IEEE, 2012.

[17] Courtois, P., Heymans, F., and Parnas, D., "Concurrent control with readers and writers," *Communications of the ACM*, vol. 14, no. 10, pp. 667–668, 1971.

[18] Couvreur, J., Encrenaz, E., Paviot-Adet, E., Poitrenaud, D., and Wacrenier, P., "Data decision diagrams for Petri net analysis," *Application and Theory of Petri Nets 2002*, pp. 129–158, 2002.

[19] Couvreur, J. and Thierry-Mieg, Y., "Hierarchical decision diagrams to exploit model structure," *Formal Techniques for Networked and Distributed Systems-FORTE*, pp. 443–457, 2005.

[20] Dickson, L., "Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors," *American Journal of Mathematics*, vol. 35, no. 4, pp. 413–422, 1913.

[21] Dijkstra, E. W., "Cooperating sequential processes," tech. rep., Technological University, Eindhoven, Netherlands, 1965.

[22] Ezpeleta, J., Colom, J., and Martinez, J., "A Petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 2, pp. 173–184, 1995.

[23] Ezpeleta, J., Tricas, F., Garcia-Valles, F., and Colom, J. M., "A Banker's solution for deadlock avoidance in FMS with flexible routing and multi-resource states," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 621–625, 2002.

[24] Fanti, M. P., Maione, B., Mascolo, S., and Turchiano, B., "Event-based feedback control for deadlock avoidance in flexible production systems," *IEEE Transactions on Robotics and Automation*, vol. 13, pp. 347–363, 1997.

[25] Feng, L. and Wonham, W., "TCT: A computation tool for supervisory control synthesis," in *Proceedings of the 8th International Workshop on Discrete Event Systems*, pp. 388–389, IEEE, 2006.

[26] Ghaffari, A., Rezg, N., and Xie, X., "Design of a live and maximally permissive Petri net controller using the theory of regions," *IEEE Transactions on Robotics and Automation*, vol. 19, pp. 137–141, 2003.

[27] Giua, A., DiCesare, F., and Silva, M., "Generalized mutual exclusion constraints on nets with uncontrollable transitions," in *Proceedings of the 1992 IEEE International Conference on Systems, Man and Cybernetics*, pp. 974–979.

[28] Gohari, P. and Wonham, M. W., "On the complexity of the supervisory control design in the RW framework," *IEEE Transactions on SMC – Part B*, vol. 30, pp. 643–652, 2000.

[29] Gold, E. M., "Deadlock prediction: Easy and difficult cases," *SIAM Journal of Computing*, vol. 7, pp. 320–336, 1978.

[30] Habermann, A. N., "Prevention of system deadlocks," *Communications of the ACM*, vol. 12, pp. 373–377, 1969.

[31] Havender, J. W., "Avoiding deadlock in multi-tasking systems," *IBM Systems Journal*, vol. 2, pp. 74–84, 1968.

[32] Holt, R. D., "Some deadlock properties of computer systems," *ACM Computing Surveys*, vol. 4, pp. 179–196, 1972.

[33] Hopcroft, J. E. and Ullman, J. D., *Introduction to Automata Theory, Languages and Computation*. Reading, MA: Addison-Wesley, 1979.

[34] Hsieh, F. S. and Chang, S. C., "Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 10, pp. 196–209, 1994.

[35] Huang, Y., Jeng, M., Xie, X., and Chung, D., "Siphon-based deadlock prevention policy for flexible manufacturing systems," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 36, no. 6, pp. 1248–1256, 2006.

[36] Iordache, M. and Antsaklis, P., *Supervisory Control of Concurrent Systems: A Petri Net Structural Approach*. Birkhauser, 2006.

[37] Jeng, M., Xie, X., and Peng, M. Y., "Process nets with resources for manufacturing modeling and their analysis," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 875–889, 2002.

[38] Kelly, T., Wang, Y., Lafortune, S., and Mahlke, S., "Eliminating concurrency bugs with control engineering," *IEEE Computer*, vol. 42, no. 12, pp. 52–60, 2009.

[39] Lawley, M., Reveliotis, S., and Ferreira, P., "The application and evaluation of Banker's algorithm for deadlock-free buffer space allocation in flexible manufacturing systems," *International Journal of Flexible Manufacturing Systems*, vol. 10, pp. 73–100, 1998.

[40] Lawley, M., Reveliotis, S., and Ferreira, P., "A correct and scalable deadlock avoidance policy for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 796–809, 1998.

[41] Lawley, M. A. and Reveliotis, S. A., "Deadlock avoidance for sequential resource allocation systems: hard and easy cases," *International Journal of FMS*, vol. 13, pp. 385–404, 2001.

[42] Li, Z., Zhou, M., and Wu, N., "A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems," *IEEE Transactions Systems, Man and Cybernetics – Part C: Applications and Reviews*, vol. 38, pp. 173–188, 2008.

[43] Li, Z. and Zhou, M., *Deadlock resolution in automated manufacturing systems: a novel Petri net approach.* Springer Verlag, 2009.

[44] Li, Z., Hu, H., and Wang, A., "Design of liveness-enforcing supervisors for flexible manufacturing systems using Petri nets," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 37, no. 4, pp. 517–526, 2007.

[45] Li, Z. and Zhou, M., "Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 34, no. 1, pp. 38–51, 2004.

[46] Li, Z. and Zhou, M., "Clarifications on the definitions of elementary siphons in Petri nets," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 36, no. 6, pp. 1227–1229, 2006.

[47] Li, Z. and Zhou, M., "Control of elementary and dependent siphons in Petri nets and their application," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 38, no. 1, pp. 133–148, 2008.

[48] Ma, C. and Wonham, W., "Nonblocking supervisory control of state tree structures," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 782–793, 2006.

[49] Miner, A. and Ciardo, G., "Efficient reachability set generation and storage using decision diagrams," *Application and Theory of Petri Nets*, pp. 691–691, 1999.

[50] Miremadi, S., Akesson, K., and Lennartson, B., "Extraction and representation of a supervisor using guards in extended finite automata," in *Proceedings of the 9th International Workshop on Discrete Event Systems*.

[51] Miremadi, S., Åkesson, K., and Lennartson, B., "A BDD-based approach for modeling plant and supervisor by extended finite automata," tech. rep., Chalmers University of Technology, 2010.

[52] Miremadi, S., Akesson, K., and Lennartson, B., "Symbolic computation of reduced guards in supervisory control," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 754–765, 2011.

[53] Moody, J. O. and Antsaklis, P. J., *Supervisory Control of Discrete Event Systems using Petri nets*. Boston, MA: Kluwer Academic Pub., 1998.

[54] Murata, T., "Petri nets: Properties, analysis and applications," in *Proceedings of the IEEE*, vol. 77, pp. 541–580, 1989.

[55] Nazeem, A. and Reveliotis, S., "Designing maximally permissive deadlock avoidance policies for sequential resource allocation systems through classification theory," in *Proceedings of the 2011 IEEE Conference on Automation Science and Engineering*.

[56] Nazeem, A. and Reveliotis, S., "An efficient algorithm for the enumeration of the minimal unsafe states in complex resource allocation systems," in *Proceedings of the 2012 IEEE Conference on Automation Science and Engineering (Accepted)*.

[57] Nazeem, A. and Reveliotis, S., "A practical approach to the design of maximally permissive liveness-enforcing supervisors for complex resource allocation systems," in *Proceedings of the 2010 IEEE Conference on Automation Science and Engineering*, pp. 451–458.

[58] Nazeem, A. and Reveliotis, S., "A practical approach for maximally permissive liveness-enforcing supervision of complex resource allocation systems," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 4, pp. 766–779, 2011.

[59] Nazeem, A. and Reveliotis, S., "Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory: the non-linear case," *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1670–1684, 2012.

[60] Nazeem, A. and Reveliotis, S., "Maximally permissive deadlock avoidance for resource allocation systems with r/w-locks," in *Proceedings of the 11th International Workshop on Discrete Event Systems (Accepted)*, IEEE, 2012.

[61] Nazeem, A., Reveliotis, S., Wang, Y., and Lafortune, S., "Optimal deadlock avoidance for complex resource allocation systems through classification theory," in *Proceedings of the 10th International Workshop on Discrete Event Systems*, IEEE, 2010.

[62] Nazeem, A., Reveliotis, S., Wang, Y., and Lafortune, S., "Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory: the linear case," *IEEE Transactions on Automatic Control*, vol. 56, no. 8, pp. 1818–1822, 2011.

[63] Nilsson, N. J., *The Mathematical Foundations of Learning Machines*. San Mateo, CA: Morgan Kaufmann, 1990.

[64] Park, J. and Reveliotis, S., "Algebraic synthesis of efficient deadlock avoidance policies for sequential resource allocation systems," *IEEE Transactions on Robotics and Automation*, vol. 16, pp. 190–195, 2000.

[65] Park, J. and Reveliotis, S. A., "Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings," *IEEE Transactions on Automatic Control*, vol. 46, pp. 1572–1583, 2001.

[66] Pastor, E., Cortadella, J., and Roig, O., "Symbolic analysis of bounded Petri nets," *IEEE Transactions on Computers*, vol. 50, no. 5, pp. 432–448, 2001.

[67] Peterson, J. L., *Operating System Concepts*. Addison-Wesley, 1981.

[68] Ramadge, P. J. G. and Wonham, W. M., "The control of discrete event systems," in *Proceedings of the IEEE*, vol. 77, pp. 81–98, 1989.

[69] Reveliotis, S., "Algebraic deadlock avoidance policies for sequential resource allocation systems," in *Facility Logistics: Approaches and Solutions to Next Generation Challenges* (Lahmar, M., ed.), pp. 235–289, Auerbach Publications, 2007.

[70] Reveliotis, S., Roszkowska, E., and Choi, J. Y., "Generalized algebraic deadlock avoidance policies for sequential resource allocation systems," *IEEE Transactions on Automatic Control*, vol. 52, pp. 2345–2350, 2007.

[71] Reveliotis, S. A., *Structural Analysis & Control of Flexible Manufacturing Systems with a Performance Perspective*. PhD thesis, University of Illinois, Urbana, IL, 1996.

[72] Reveliotis, S. A., *Real-time Management of Resource Allocation Systems: A Discrete Event Systems Approach*. NY, NY: Springer, 2005.

[73] Reveliotis, S. A., "Implicit siphon control and its role in the liveness enforcing supervision of sequential resource allocation systems," *IEEE Transactions on SMC: Part A*, vol. 37, pp. 319–328, 2007.

[74] Reveliotis, S. A. and Ferreira, P. M., "Deadlock avoidance policies for automated manufacturing cells," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 845–857, 1996.

[75] Reveliotis, S. A., Lawley, M. A., and Ferreira, P. M., "Polynomial complexity deadlock avoidance policies for sequential resource allocation systems," *IEEE Transactions on Automatic Control*, vol. 42, pp. 1344–1357, 1997.

[76] Reveliotis, S. A. and Nazeem, A., "Optimal linear separation of the safe and unsafe subspaces of sequential RAS as a set-covering problem: algorithmic procedures and geometric insights," tech. rep., School of Industrial & Systems Eng., Georgia Tech, 2012.

[77] Ricker, L., Lafortune, S., and Gene, S., "DESUMA: A tool integrating Giddes and Umdes," in *Proceedings of the 8th International Workshop on Discrete Event Systems*, pp. 392–393, IEEE, 2006.

[78] Thierry-Mieg, Y., Poitrenaud, D., Hamez, A., and Kordon, F., "Hierarchical set decision diagrams and regular models," *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 1–15, 2009.

[79] Tricas, F., García-Vallés, F., Colom, J., and Ezpeleta, J., "A Petri net structure–based deadlock prevention solution for sequential resource allocation systems," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 271–277.

[80] Tsitsiklis, J., "On the control of discrete-event dynamical systems," *Mathematics of Control, Signal and Systems*, vol. 2, pp. 95–107, 1989.

[81] Uzam, M., "An optimal deadlock prevention policy for flexible manufacturing systems using Petri net models with resources and the theory of regions," *International Journal of Advanced Manufacturing Technology*, vol. 19, pp. 192–208, 2002.

[82] Vahidi, A., Fabian, M., and Lennartson, B., "Efficient supervisory synthesis of large systems," *Control engineering practice*, vol. 14, no. 10, pp. 1157–1167, 2006.

[83] Valk, R. and Jantzen, M., "The residue of vector sets with applications to decidability problems in Petri nets," *Acta Informatica*, vol. 21, no. 6, pp. 643–674, 1985.

[84] Valmari, A., "Stubborn sets for reduced state space generation," *Advances in Petri Nets*, pp. 491–515, 1991.

[85] Vapnik, V., *The nature of statistical learning theory*. Springer-Verlag New York Inc, 2000.

[86] Varpaaniemi, K., "Efficient detection of deadlocks in Petri nets," *Helsinki University of Technology, Digital Systems Laboratory Report A*, vol. 26.

[87] Vazirani, V., *Approximation Algorithms*. NY,NY: Springer, 2003.

[88] Viswanadham, N., Narahari, Y., and Johnson, T., "Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models," *IEEE Transactions on Robotics & Automation*, vol. 6, no. 6, pp. 713–723, 1990.

[89] Wang, Y., Kelly, T., Kudlur, M., Lafortune, S., and Mahlke, S., "Gadara: Dynamic deadlock avoidance for multithreaded programs.," in *Proceedings of the 8th Usenix Symposium on Operating Systems Design and Implementation*, pp. 281–294, USENIX Association, 2008.

[90] Wang, Y., Lafortune, S., Kelly, T., Kudlur, M., and Mahlke, S., "The theory of deadlock avoidance via discrete control," in *Proceedings of the 36th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, (New York, NY, USA), pp. 252–263, ACM, 2009.

[91] Wang, Y., Liao, H., Reveliotis, S., Kelly, T., Mahlke, S., and Lafortune, S., "Gadara nets: Modeling and analyzing lock allocation for deadlock avoidance in multithreaded software," in *Proceedings of the 48th IEEE Conference on Decision and Control*, pp. 4971 – 4976, 2009.

[92] Wolsey, L. A., *Integer Programming*. NY, NY: John Wiley & Sons, 1998.

[93] Xing, K. Y., Hu, B. S., and Chen, H. X., "Deadlock avoidance policy for Petri net modeling of flexible manufacturing systems with shared resources," *IEEE Transactions on Automatic Control*, vol. 41, pp. 289–295, 1996.

[94] Zhang, Z. and Wonham, W., "Synthesis and control of discrete-event systems," 2002.

[95] Zhou, M. and Fanti (editors), M. P., *Deadlock Resolution in Computer-Integrated Systems*. Singapore: Marcel Dekker, Inc., 2004.

VITA

Ahmed Nazeem was born in Cairo, Egypt on July 3, 1982. He received his B.S. in Computer Engineering from Cairo University in 2004, and a M.S. in Industrial Engineering from Georgia Institute of Technology in 2009. His research interest is in the area of discrete event systems theory and applications. In 2010, Ahmed was the recipient of the Thorlabs Best Student Paper Award for the IEEE Conference on Automation Science and Engineering.